UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE CIÊNCIAS E TECNOLOGIA
PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

UNIDADE ARITMÉTICA BÁSICA PARA TRANSDUÇÃO DIGITAL DE PARÂMETROS DE POTÊNCIA

RÔMULO PIRES COELHO FERREIRA

CAMPINA GRANDE - PB
DEZEMBRO - 1993

RÔMULO PIRES COELHO FERREIRA

UNIDADE ARITMÉTICA BÁSICA PARA TRANSDUÇÃO DIGITAL DE PARÂMETROS DE POTÊNCIA

Dissertação apresentada ao curso de MESTRADO EM ENGENHARIA ELÉTRICA da Universidade Federal da Paraíba, em cumprimento às exigências para obtenção do GRAU DE MESTRE.

ÁREA DE CONCENTRAÇÃO: PROCESSAMENTO DA INFORMAÇÃO

ANTONIO CARLOS CAVALCANTI - Dr. - Ing - UFPb
Orientador

MISAEL ELIAS DE MORAES - Dr. - Ing - UFPb
Orientador

Campina Grande - PB
dezembro - 1993

F383u Ferreira, Rômulo Pires Coelho.
Unidade aritmética básica para transdução digital de parâmetros de potência / Rômulo Pires Coelho Ferreira. - Campina Grande, 1993.
115 f.

Dissertação (Mestrado em Engenharia Elétrica) - Universidade Federal da Paraíba, Centro de Ciências e Tecnologia, 1993.
"Orientação : Prof. Dr. Antonio Carlos Cavalcanti, Prof. Dr. Misael Elias de Moraes".
Referências.

1. Potência Elétrica. 2. Transdutores de Parâmetros de Potência. 3. Transdução Digital. 4. Dissertação - Engenharia Elétrica. I. Cavalcanti, Antonio Carlos. II. Moraes, Misael Elias de. III. Universidade Federal da Paraíba - Campina Grande (PB). IV. Título

CDU 621.3.026(043)

UNIDADE ARITMÉTICA BÁSICA PARA TRANSDUÇÃO DIGITAL DE PARÂMETROS DE POTÊNCIA

RÔMULO PIRES COELHO FERREIRA

DISSERTAÇÃO APROVADA EM: 20/12/1993

ANTONIO CARLOS CAVALCANTI - Dr. - Ing - UFPb
Orientador

MISAEL ELIAS DE MORAES - Dr. - Ing - UFPb
Orientador

JOÃO MARQUES DE CARVALHO - Ph.D - UFPb
Componente da Banca

HAMILTON SOARES DA SILVA - Mst - UFPb
Componente da Banca

Campina Grande - PB
dezembro - 1993

AGRADECIMENTOS

Aos professores Antonio Carlos Cavalcanti e Misael Elias de Moraes, pela valiosa orientação dada a este trabalho.

Aos professores Hamilton Soares e José Antonio, pela colaboração dada para a conclusão deste trabalho.

Ao professor Gurdip Sing Deep, pela orientação acadêmica dada.

Ao professor William Ferreira Giozza, pela oportunidade dada à realização de pesquisas na área de micro-eletrônica.

Ao meu amigo Leônidas Francisco de Lima Júnior, pelas ajudas dadas aos problemas encontrados durante a realização deste trabalho.

Aos meus pais, Rochael e Edinete, pela confiança depositada em mim.

A minha esposa Palmira e meu filho Rômulo (Rominho), pelo apoio e confiança me dados.

Ao meu irmão Ricardo Pires, pela força dada a realização deste trabalho.

A todos os meus amigos que sempre me apoiaram na realização deste trabalho.

A DEUS

RESUMO

Os Transdutores de parâmetros de Potência disponíveis no mercado, possuem constantes de tempo da ordem de 250 ms, o que impossibilita a detecção de distúrbios que ocorram em tempos inferiores. Para solucionar este problema, foi desenvolvido no Departamento de Engenharia Elétrica da Universidade federal da Paraíba Campus II, um Transdutor Digital com constante de tempo da ordem de 16 ms, capaz de detectar tais distúrbios. O Transdutor Digital desenvolvido, fornece a um computador central, os valores das potência Ativa (P) e Reativa (Q). Os cálculos dessas potências são realizados em EPROM's e controlados por um microprocessador. Esta arquitetura só permite à realização de 160 amostras por período com uma precisão de 10 bits. Visando melhorar a performance do Transdutor Digital, foi desenvolvido um Circuito Integrado de Aplicação Específica (ASIC) que realizará todas as funções aritméticas deste. Com este ASIC, a Unidade Aritmética Básica (UAB), será possível realizar 660 amostras por período com uma precisão de 16 bits. A UAB realizará operações de Soma, Subtração, Multiplicação e Divisão em 32 bits. A comunicação se dá através de um barramento único de entrada/saída de 16 bits. O sistema utilizado na implementação da UAB foi o ALLIANCE, ferramenta desenvolvida pelo laboratório MASI/PARIS 6.

SUMÁRIO

LISTA DE FIGURAS E TABELAS

FIGURAS

CAPÍTULO I

CAPÍTULO II

CAPÍTULO III

CAPÍTULO IV

CAPÍTULO V

CAPÍTULO VI

TABELAS

CAPÍTULO II

CAPÍTULO IV

CAPÍTULO V

LISTA DE ABREVIATURAS

ASIC - Circuito Integrado de Aplicação Específica

VHDL - Linguagem de Descrição de Circuitos VLSI

VLSI - Integração em Muito Larga Escala

PMU - Projeto Multi-Usuário

VBE - Descrição Comportamental de um Circuito

VST - Descrição Estrutural de um Circuito

CAPÍTULO I

INTRODUÇÃO

Nos sistemas de distribuição de Energia Elétrica, é de suma importância a medição das potências Ativa (P) e Reativa (Q), pois é com seu conhecimento que se pode determinar o custo e a utilização do sistema.

Com as constantes medições dessas grandezas, é possível acompanhar a taxa de crescimento do consumo de Energia Elétrica. Com isso, pode-se por exemplo, fornecer informações que auxiliam os estudos de expansão do sistema de distribuição. Pode-se ainda, detectar anormalidades, possibilitando a atuação de um controle, manual ou automático, de proteção do sistema.

As medições dos parâmetros de potência são feitas através de Transdutores específicos para sistemas de distribuição de Energia Elétrica de alta tensão. Estes instrumentos estão dispostos em paineis de controle localizadas em Subestações Rebaixadoras, e são manipulados por um operador. Normalmente são instrumentos analógicos que utilizam um galvanômetro para o fornecimento dos resultados. Este sistema apresenta duas limitações básicas: exigência da permanência absoluta e atenção do operador diante dos instrumentos e impossibilidade de detecção de distúrbios que ocorram em tempos curtos, pois os galvanômetros não respondem a variações rápidas. Devido as constantes de tempo intrínseca a estes instrumentos.

Para resolver estes problemas, surgiram os Transdutores Digitais de Parâmetros de Potência, que além de detectarem os distúrbios de curta duração, ainda permitem a automatização dos sistemas de controle das subestações pois, por serem instrumentos digitais, podem ser facilmente conectados a computadores.

Os Transdutores Digitais comerciais possuem constantes de tempo da ordem de 250 ms, o que torna possível a detecção de vários distúrbios que podem ocorrer nos sistemas de distribuição. Mas devido ao constante crescimento destes sistemas, os distúrbios ocorridos em tempos inferiores a 250 ms, como por exemplo, as reações Eletro-mecânicas que são da ordem de 100 ms, precisam ser medidos e controlados para que o sistema funcione normalmente. Assim, com os transdutores comerciais não se pode medir tais distúrbios, uma vez que sua constante de

tempo (250 ms) é maior que a duração dos distúrbios que se pretende medir (100 ms, por exemplo), tornando-se uma limitação deste dispositivo.

Na solução dessa limitação, foi desenvolvido no Departamento de Engenharia Elétrica (DEE) da Universidade Federal da Paraíba (UFPb) [MOR90], um Transdutor Digital de Parâmetros de Potência com constante de tempo da ordem de 16 ms, capaz de realizar aproximadamente, 160 amostras por período com uma precisão de 10 bits. O sistema está conectado a um computador central (hospedeiro) que faz todo o controle e leitura dos Transdutores utilizados. Este Transdutor está sendo utilizado pela Companhia Hidro-Elétrica do São Francisco (CHESF) apresentando resultados satisfatórios.

A distribuição de Energia Elétrica é feita em redes trifásicas, implicando nas medições das potências Ativa e Reativa em cada fase. Como o Transdutor Digital desenvolvido é monofásico, tem-se que utilizar um conjunto de três Transdutores por rede (Transformador). Assim, em um sistema complexo, o número de Transdutores necessários é muito grande, e a conexão com o computador hospedeiro é complexa (figura 1.1) pois este, além de manipular dados de 10 bits de vários Transdutores, tem que gerar sinais de controle para cada um deles e necessita muitos soquetes para conexão dos mesmos.

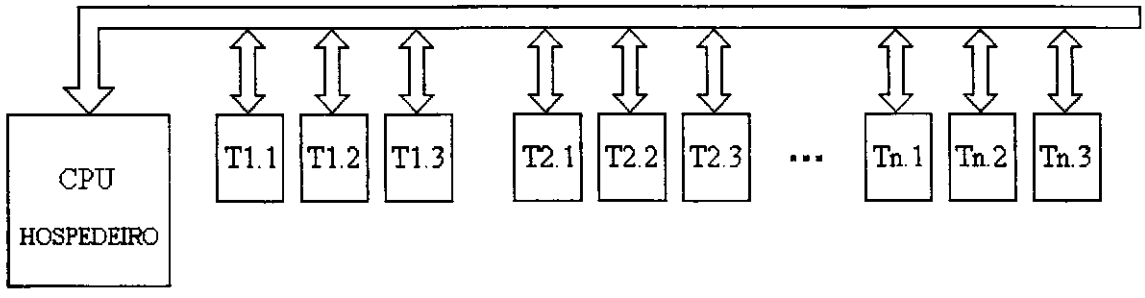


Figura 1.1 - Conexão dos Transdutores com a CPU

Visando melhorar a performance do Transdutor Digital, existe um projeto em andamento no DEE/UFPb/CAMPUS II, em colaboração com o LASIC (Laboratório de Arquitetura, Sistemas Integráveis e Circuitos) - DI/CCEN/UFPb/CAMPUS I, de um Circuito Integrado de Aplicação Específica (ASIC) que fará todas as funções digitais do Transdutor. Com este ASIC, será possível realizar aproximadamente 660 amostras por período com precisão de 16 bits. Por se tratar de um dispositivo de pequenas dimensões, em uma única placa de Circuito Impresso, será possível, no mínimo, montar os três Transdutores necessários por rede de distribuição, assim, as conexões com o computador hospedeiro serão bastante simplificadas (figura 1.2).

O projeto de desenvolvimento do ASIC que realiza as funções digitais do Transdutor, foi dividido em dois grandes blocos: Uma Unidade Aritmética que realizará todos os cálculos do Transdutor, e uma Unidade de Controle que fará a seleção das operações aritméticas, o carregamento dos operandos na Unidade Aritmética e o interfaceamento do Transdutor com o computador hospedeiro.

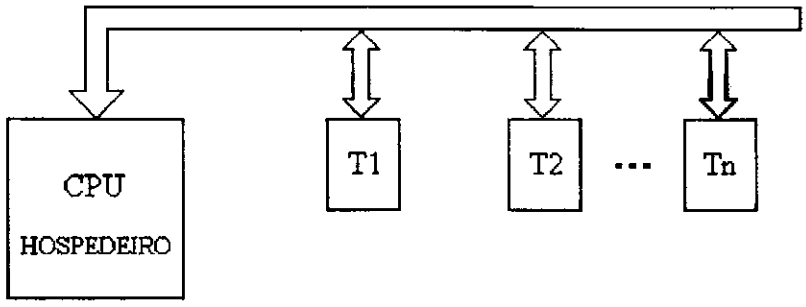


Figura 1.2 - Nova Conexão dos Transdutores com a CPU

No presente trabalho, é apresentada uma arquitetura para a Unidade Aritmética do Transdutor Digital, que por realizar as operações de Adição, Subtração, Multiplicação e Divisão foi denominada Unidade Aritmética Básica (UAB). A UAB realiza operações com 32 bits. Para isso foi utilizado o sistema ALLIANCE [MAS93] para a concepção deste circuito integrado. A tecnologia utilizada foi a "STANDARD CELLS" CMOS de 1,2 μm com dois níveis de metal, respeitando os parâmetros estabelecidos pelo Projeto Multi-Usuário (PMU). A metodologia de projeto foi a "TOP-DOWN" utilizando "VHDL" (VLSI; Hardware Description Language).

O capítulo 2 apresenta o projeto original do Transdutor Digital. São mostradas as operações aritméticas realizadas para o fornecimento das potências e a arquitetura implementada.

O capítulo 3 apresenta o sistema ALLIANCE, apresentando suas ferramentas e sua utilização. Também serão apresentadas a metodologia de projeto "TOP-DOWN" utilizando estas ferramentas e a linguagem de descrição de circuitos "VLSI" (Very Large Scale Intagration), "VHDL".

O capítulo 4 mostra a metodologia adotada no projeto da UAB, bem como suas bases, ou seja, o funcionamento dos algoritmos implementados na UAB.

O capítulo 5 mostra a implementação da UAB. Partindo das bases, definidas no capítulo 4, os circuitos e algoritmos são unificados em um módulo de operações aritméticas.

CAPÍTULO II

O TRANSDUTOR DIGITAL

Este capítulo apresenta o Transdutor Digital de Parâmetros de Potência desenvolvido no DEE/UFPb/CAMPUS II [MOR90]. Serão apresentados os cálculos realizados pelo Transdutor, a metodologia adotada no projeto e a nova estrutura incluindo o ASIC proposto.

2.1 - As Funções do Transdutor Digital

A função do Transdutor Digital é calcular os valores das potências Ativa (P) e Reativa (Q), fornecer o sinal da potência Reativa, indicando se a carga é capacitiva ou indutiva e o valor da frequência da rede. Para detecção do início do ciclo da rede, o Transdutor recebe um sinal de passagem por zero (Z0). A figura 2.1 mostra os sinais de interface do Transdutor.

Figura 2.1 - Interface do Transdutor Digital

Para os cálculos das potências, o Transdutor deve resolver as seguintes equações:

$$P = \frac{1}{T}\int_0^T v(t).i(t).dt \qquad (2.1)$$

$$Q = \sqrt{S^2 - P^2} \qquad (2.2)$$

onde, $$S = V_{RMS} \cdot I_{RMS} \qquad (2.3)$$

$$V_{RMS} = \sqrt{\frac{1}{T}\int_0^T v^2(t).dt} \qquad (2.4)$$

$$I_{RMS} = \sqrt{\frac{1}{T}\int_0^T i^2(t).dt} \qquad (2.5)$$

A figura 2.2 mostra o diagrama (triângulo das potências) que relaciona as potências Ativa (P), Reativa (Q) e Aparente(S).

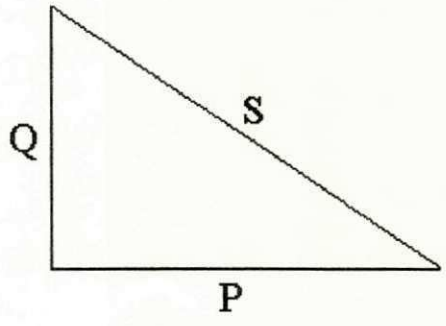


Figura 2.2 - Triângulo das Potencias

Na discretização, as integrais são realizadas com somas sucessivas, assim o período T é substituído pelo número dessas somas, ou seja, pelo número de amostras N. Assim, discretizando as equações 2.1, 2.4 e 2.5 ficam:

$$P = \frac{1}{N}\sum_{t=0}^{N} v(t).i(t) \qquad (2.6)$$

$$V_{RMS} = \sqrt{\frac{1}{N}\sum_{t=0}^{N} v^2(t)} \qquad (2.7)$$

$$I_{RMS} = \sqrt{\frac{1}{N}\sum_{t=0}^{N} i^2(t)} \qquad (2.8)$$

A frequência de amostragem é fixa, logo o tempo necessário a realização de uma amostra é fixo, assim conhecendo-se o número de amostras N, é possível calcular a frequência da rede. Para se saber o valor de N, um contador de eventos é disparado quando um pulso é registrado em Z0, indicando que houve uma passagem por zero (figura 2.3), de tensão ou de corrente. A seguir, ao final de cada amostra este contador é incrementado até uma outra passagem por zero.

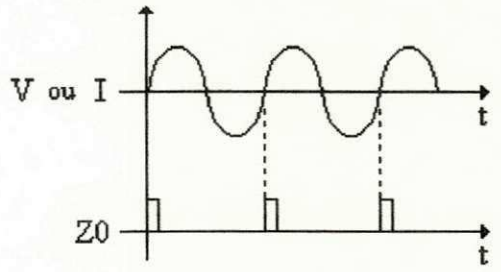


Figura 2.3 - Sinal de Passagem por Zero

O sinal da carga é obtido a partir de um comparador de fase; assim, se a tensão estiver adiantada em relação a corrente, a carga é indutiva (sinal -), caso contrário é capacitiva (sinal +).

2.2 - Metodologia de Projeto

A metodologia adotada no projeto do Transdutor Digital, pode ser expressada em quatro etapas importantes:

- Conversão Analógico/Digital,
- Operações Aritméticas,

- Fornecimento dos Resultados e
- Controle.

Por se tratar de um circuito digital, os valores de tensão e corrente devem ser convertidos de Analógicos para Digitais através de conversores A/D. Para as operações aritméticas foi utilizada a técnica do uso de EPROM's como tabela (ver seção 2.2.1). Os resultados das operações do Transdutor são fornecidos em barramento de 10 bits. O controle do Transdutor é feito por um microprocessador, que faz o interfaceamanto com a CPU central e gera os complementos dos endereços às EPROM's de forma a obter os valores das operações aritméticas. O diagrama de blocos do Transdutor é mostrado na figura 2.4.

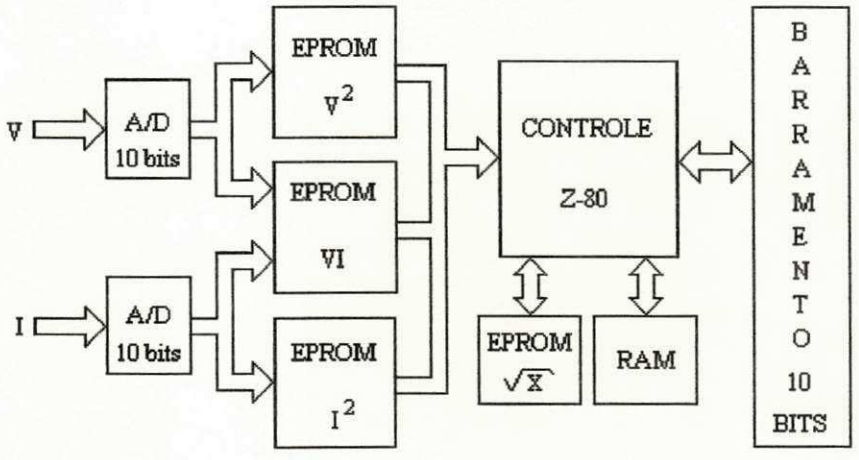


Figura 2.4 - Diagrama de Blocos do Transdutor Digital

2.2.1 - Uso de EPROM como Tabela

No Transdutor Digital, a técnica de utilizar as EPROM's como tabela [TAU82], foi adotada para a implementação das operações aritméticas de quadrado (v^2 e i^2), produto (v.i) e raiz quadrada.

Nessa técnica, os valores tabelados são guardados nas locações da EPROM. Normalmente os dados armazenados têm relação com o endereço para facilitar a manipulação dos valores desejados. Para exemplificar melhor esta técnica, será mostrado a implementação de uma operação de quadrado (a mesma utilizada no Transdutor) com operando de três bits. A tabela da verdade do quadrador é a seguinte:

OPERANDO (O)			QUADRADO (Q)							
O_2	O_1	O_0	Q_7	Q_6	Q_5	Q_4	Q_3	Q_2	Q_1	Q_0
0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	1
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	1
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	0	1	1	0	0	1
1	1	0	0	0	1	0	0	1	0	0
1	1	1	0	0	1	1	0	0	0	1

Tabela 2.1 - Tabela da Verdade do Quadrador de três bits

Para implementar a função de quadrado de três bits em uma tabela, precisamos de uma EPROM com oito locações de oito bits (8x8). Assim, o endereço é o operando e o conteúdo da locação é o quadrado do operando (figura 2.5).

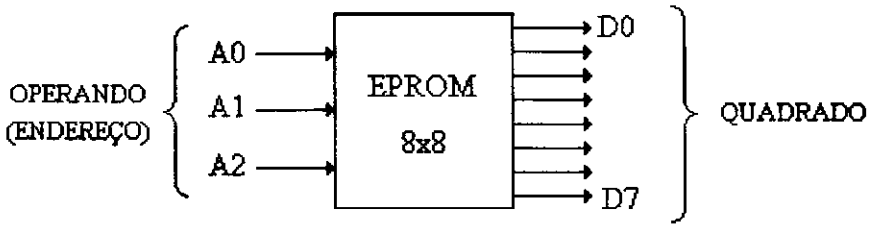


Figura 2.5 - Representação de uma EPROM como Tabela

Desta mesma forma pode-se implementar funções como o produto, raiz quadrada **etc.**

No exemplo ilustrado, a operação é realizada com facilidade porque o resultado cabe em uma locação de memória. Isto não acontece no projeto do Transdutor, pois como os operandos são de 10 bits, as locações deveriam ser de vinte bits. Neste caso, utiliza-se a técnica de fragmentar os vinte bits em blocos de oito bits (bytes) e alocar em posições de memória na sequência do menos significativo ao mais significativo. Assim, o operando compõe a parte mais significativa do endereço, e a parte menos significativa fica como seleção dos bytes do resultado. Para ilustrar esta técnica, a figura 2.6 mostra como fica armazenado o quadrado de "1001101110" (622D), onde este número é o endereço.

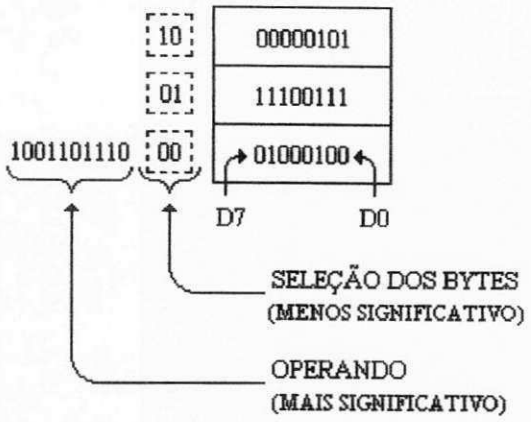


Figura 2.6 - Operação de Quadrado Ocupando Várias Locações

No Transdutor Digital, o operando (parte mais significativa do endereço) é o valor na saída do A/D. A figura 2.7 mostra a ligação do A/D com a EPROM.

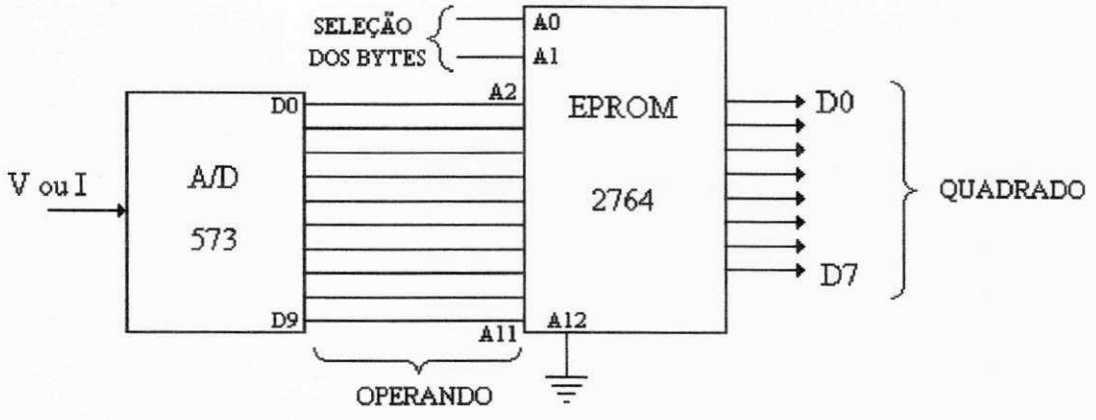


Figura 2.7 - Ligação do A/D com a EPROM

2.3 - Características do Transdutor Digital

As características do Transdutor são:

- Precisão de 10 bits,
- Constante de tempo de 16,67 ms,
- Realização de 160 amostras por período e
- Frequência de clock de 6MHz.

Com a arquitetura adotada no projeto, só é possível realizar 160 amostras por período, pois o microprocessador controla todas as operações do Transdutor. E ela quem envia os complementos dos endereços das EPROM's de forma a obter os produtos, quadrados e raizes quadradas. Depois, faz todo o interfaceamento do Transdutor com a CPU central. Todo este processamento demanda muito tempo, de forma a limitar a capacidade de realização de um número maior de amostras.

2.4 - O Uso de um ASIC no Transdutor

A inclusão de um ASIC no projeto do Transdutor Digital é de grande importância, pois suas características (precisão e número de amostras) serão melhores. A figura 2.8 mostra o diagrama de blocos do Transdutor usando o ASIC.

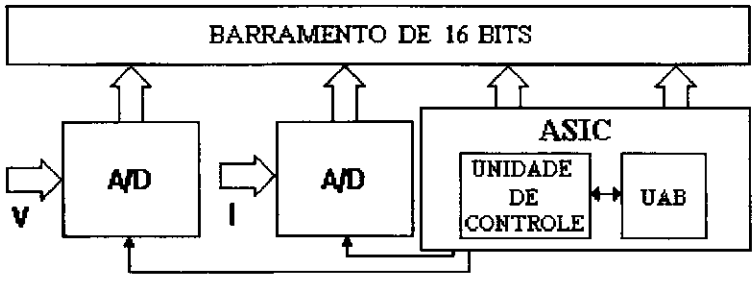


Figura 2.8 - Transdutor Digital Usando um ASIC

Com esta arquitetura, as características do Transdutor Digital são:

- Precisão de 32 bits,
- Constante de tempo de 16,67 ms,
- Realização de 660 amostras por período e
- Frequência de clock de 33MHz.

Com o projeto desse ASIC, o Transdutor terá uma redução significativa no número de chips na placa, possibilitando por exemplo, a confecção dos três módulos em uma única placa, satisfazendo o diagrama da figura 1.2.

Conforme a figura 2.8, o ASIC proposto possui dois grandes blocos: Uma Unidade Aritmética, assunto deste trabalho (ver capítulo IV) e uma Unidade de Controle, que também está em desenvolvimento no DEE/UFPb/CAMPUS II em cooperação com o LASIC/DI/CCEN/UFPb/CAMPUS I.

CAPÍTULO III

O ALLIANCE

O sistema ALLIANCE é um conjunto de ferramentas de CAD para projetos VLSI CMOS, desenvolvido no laboratório MASI/CAO-VLSI da Universidade Pierre et Marie Curie (UPMC - Paris 6) em Paris.

Para o desenvolvimento de projetos VLSI, o ALLIANCE utiliza um subconjunto da linguagem "VHDL" (VLSI Hardware Description Language) padrão IEEE [IEEE88].

Composto pelo "VHDL" comportamental (VHDL Behavior - **vbe**) [MAS93c] e estrutural (VHDL Structural - **vst**) [MAS93c] que serão apresentados na seção 3.1. As etapas de projeto (lógico, fisico e verificação) e as ferramentas do sistema ALLIANCE serão apresentadas na seção 3.2. A seção 3.3 apresenta a metodologia de projeto "TOP-DOWN" utilizando estas ferramentas.

3.1 - VHDL

O "VHDL" é uma linguagem de descrição de circuitos "VLSI" (Very Large Scale Integration) desenvolvido pelo "INSTITUTE OF ELECTRIC AND ELECTRONIC ENGINEERS" (IEEE). O ALLIANCE utiliza o "VHDL" "Behavioral" e "Structural" para o desenvolvimento de projetos. A descrição comportamental (**vbe**) de um circuito, utiliza operadores lógicos (not, and, or, xor, etc), Latches, Flip-Flops, Mux etc, de forma a implementar o projeto lógico. Já a descrição estrutural (**vst**) apresenta as ligações entre os blocos componentes de um circuito ("NETLIST").

Para exemplificar o "VHDL" **vbe** e **vst**, considere um circuito que implemente a função lógica

$$F = \overline{A} + B$$

a descrição **vbe** deste circuito será:

```
ENTITY exemplo IS
        PORT (
                A  : IN BIT;
                B  : IN BIT;
                F  : OUT BIT
              );
END exemplo;

ARCHITECTURE comportamental OF exemplo IS

BEGIN
        F <= (NOT A) OR B;
END comportamental;
```

A descrição **vst** deste circuito, conhecendo-se as descrições vbe dos componentes "inversor" e "ou", será:

```
ENTITY exemplo IS
        PORT (
                A  : IN BIT;
                B  : IN BIT;
                F  : OUT BIT
              );
END exemplo;

ARCHITECTURE estrutural OF exemplo IS;

        COMPONENT inversor;

                PORT (  I   : IN BIT;
                        O   : OUT BIT  );

        END COMPONENT;

        COMPONENT ou;

                PORT (  I1  : IN BIT;
                        I2  : IN BIT;
                        O   : OUT BIT  );

        END COMPONENT;

SIGNAL not_A : BIT

BEGIN
        inv : inversor
                PORT MAP (  I   => A,
                            O   => not_A  );

        or1 : ou
                PORT MAP (  I1  => not_A,
                            I2  => B,
                            O   => F  );
END estrutural;
```

Observa-se que a descrição **vbe** apenas implementa as funções do circuito e o **vst** descreve as ligações entre seus componentes. A figura 3.1 mostra a representação da descrição **vbe** de um circuito e a figura 3.2 a descrição **vst**. Observa-se ainda, que a descrição estrutural pressupõe a descrição comportamental de cada bloco utilizado, pois descreve apenas interligações ("NETLIST") de sinais e blocos.

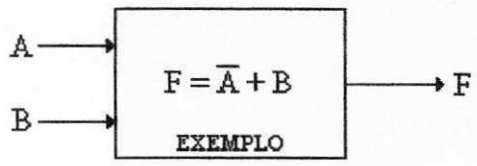


Figura 3.1 - Representação da Descrição **vbe** de um Circuito

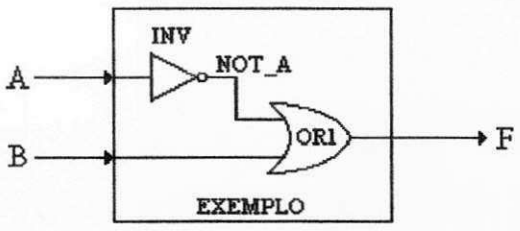


Figura 3.2 - Representação da Descrição **vst** de um Circuito

3.2 - As Ferramentas do Sistema ALLIANCE

As ferramentas do sistema ALLIANCE foram desenvolvidas de forma a se comunicarem, além de poderem ser utilizadas independentemente.

O conjunto das ferramentas do ALLIANCE será mostrado a seguir com sua descrição:

ASIMUT

O ASIMUT [MAS93b] é um simulador lógico "VHDL". Suporta as descrições comportamentais (behavioral) e estruturais (structural), que formam um subconjunto "VHDL".

SCLIB

A SCLIB [MAS93a] é uma biblioteca "STANDARD-CELLS" CMOS, que contém mais de 60 células diferentes. Para cada célula existe um modelo "VHDL"

correspondente (descrição), um esquemático ao nível de transistores (descrição de componentes) no formato "SPICE" (Simulador Elétrico) e um arquivo de "LAYOUT" simbólico no formato ALLIANCE.

PADLIB

A PADLIB [MAS93n] é uma biblioteca de "PADS", responsável pelo interfaceamento do chip com o exterior. Inclui os "PADS" de 1.2 μm (PAD12) e 1.5 μm (PAD15), de acordo com o padrão adotado pela ES2 (European Silicon Structures).

GENLIB

O GENLIB [MAS93f] é uma linguagem procedural para descrição de "NETLIST" e/ou descrição de posicionamento de células. O GENLIB consiste num subconjunto de "C" padrão.

SCR

O SCR [MAS93d] é um posicionador e roteador para "STANDARD-CELLS".

RING

O RING [MAS93e] é o roteador da coroa de "PADS" com o coração.

S2R

O S2R [MAS93j] é um conversor de formato de "LAYOUT" simbólico para real. Os formatos de saída são "CIF" ou "GDSII".

VERSATIL

O VERSATIL [MAS93k] é um verificador de regras de projeto no nível de "LAYOUT".

LYNX

O LYNX [MAS93h] é um extrator de "NETLIST" lógica a partir do "LAYOUT". O arquivo de entrada é um "LAYOUT" simbólico e a saída pode ser no formato do "SPICE", "VHDL" ou ALLIANCE interno (al).

LVX

O LVX [MAS93g] é um comparador de "NETLIST".

DESB

O DESB [MAS93m] é um desassemblador de descrição comportamental em "VHDL" a partir de uma "NETLIST" a nível de transistor (MOS).

PROOF

O PROOF [MAS93i] faz a prova formal de duas descrições comportamentais em "VHDL".

ALC

O ALC [MAS93l] é um editor hierárquico de "LAYOUT" simbólico.

3.3 - Metodologia de Projeto

A metodologia de projeto adotada é a "TOP-DOWN". Nesta metodologia, o projeto de um circuito integrado parte de uma "caixa preta", onde se conhece apenas as entradas e saídas (relacoinadas através das equações booleanas), até atingir o ponto mais elementar do "chip". Dependendo da tecnologia, este ponto pode ser uma célula de biblioteca (por exemplo "STANDARD-CELL"), um transistor ("FULL-CUSTOM") etc.

A seguir será mostrada, em etapas, a implementação de um circuito integrado usando-se a tecnologia "STANDARD-CELLS".

ETAPA 1

Nesta etapa o circuito integrado é visto como uma "caixa preta" (figura 3.3), onde se relacionam as saídas com as entradas, ou seja, a caracterização comportamental do circuito.

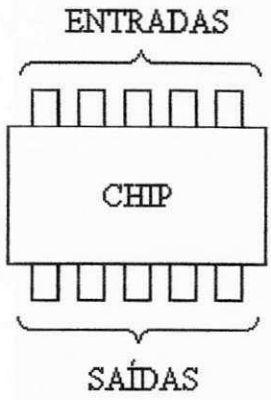


Figura 3.3 - Definição do Chip

É nesta etapa onde a descrição comportamental em "VHDL" é totalmente validada, pois é nela onde se definem todas as funções do chip. O tempo previsto para a realização desta etapa é de 75% do tempo de todo o projeto. Isto se deve ao fato de que todas as etapas seguintes estão relacionadas com esta, daí a necessidade de se validar integralmente o chip nesta etapa.

A validação do chip nesta etapa, se dá através dos vetores de teste. Estes vetores constam de combinações de valores nas entradas e seus efeitos na saída do chip. Assim, para se validar o circuito, os valores obtidos nas saídas devem coincidir com o valor esperado para cada vetor de teste. O número total de vetores de teste possíveis à validação de um circuito depende do número de entradas e de bits deste, por exemplo: um Somador de 32 bits, possui duas entradas de 32 bits, assim o número total de vetores será de 2^{64}, ou seja, $1{,}845.10^{19}$. Observa-se que este número é necessário para se testar todos os valores possíveis dos operandos, por isso o tempo estimado à realização desta etapa ser 75% (deve-se buscar o conjunto que possa testar toda a funcionalidade do circuito sem ultrapassar um número razoável de vetores, por exemplo 2.10^3). A seção 5.4 apresenta um algoritmo para escolha dos vetores de teste, a fim de se determinar o menor número de vetores que teste o circuito integralmente.

ETAPA 2

Nesta etapa define-se o coração do circuito e a coroa de "PADS" do chip. O coração é responsável pela realização de todas as funções do chip, e a coroa de "PADS" pelo interfaceamento do coração com os pinos do chip (figura 3.4).

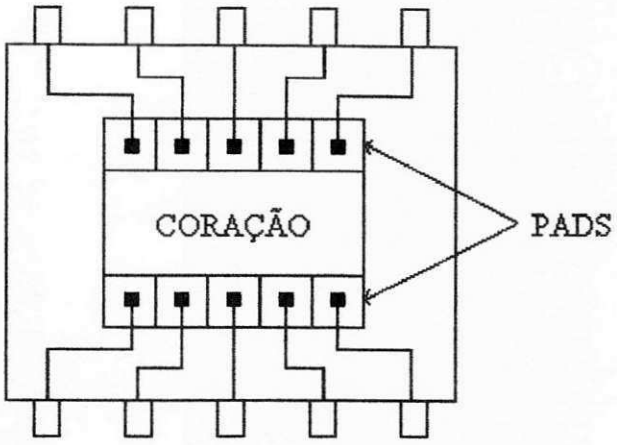


Figura 3.4 - Coração e "PADS" do Chip

Na etapa anterior o chip foi validado comportamentalmente, nesta etapa, com a divisão em coração e coroa de "PADS", o chip será validado estruturalmente; assim os vetores de teste construídos serão os mesmos.

Com a inclusão de um coração no circuito, se faz necessário sua validação comportamental e estrutural. Nesta etapa é feita a validação comportamental do coração. A diferença nas descrições do coração e do chip só diferem na inclusão dos "PADS", assim, os vetores serão ligeiramente diferentes, pois os "PADS" de saída são inversores. Para isso, basta inverter os bits de saída nos vetores de teste.

A validação dos "PADS" não se faz necessária, pois eles são integrantes de uma biblioteca de "PADS" a "PADLIB", assim eles já estão validados por definição.

ETAPA 3

Nesta etapa, o coração do chip desenvolvido na etapa 1, é subdividido em blocos principais que o compõem (figura 3.5). Isto é feito para simplificar as descrições comportamentais e estruturais de todo o chip. Assim, cada bloco é descrito e validado comportamentalmente. Em seguida são interligados de forma a realizar as funções do coração, ou seja, é feita a descrição estrutural do coração a nível de blocos.

Para a validação estrutural do coração, são utilizados os mesmos vetores de teste da etapa anterior. Os vetores de teste dos blocos são descritos individualmente para a validação (comportamental) integral de cada bloco.

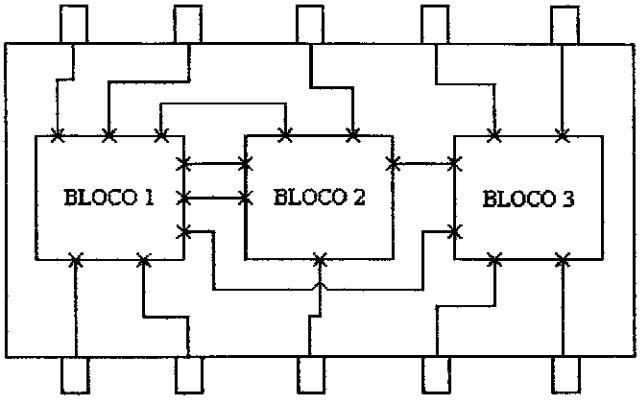


Figura 3.5 - Divisão do Coração em Blocos

ETAPA 4

Nesta etapa os blocos desenvolvidos na etapa anterior, podem ser novamente divididos em sub-blocos, dependendo da complexidade do circuito. Neste caso, esta etapa será idêntica a anterior, cada sub-bloco será descrito comportamentalmente e validados individualmente. Os blocos da etapa anterior podem ser validados estruturalmente e validados com os vetores de teste correspondentes.

Como esta etapa depende da complexidade do circuito, ela pode ou não existir. No caso negativo, pula-se para a etapa 5.

ETAPA 5

Nesta etapa os blocos, ou sub-blocos, desenvolvidos nas etapas 3 ou 4, são subdivididos em slices (fatias operativas de 1 bit), ver figura 3.6. Com isso os slices são validados e interligados de forma a realizarem as funções dos blocos, ou seja, é feita a descrição estrutural dos blocos. Ainda nesta etapa, é feita a descrição estrutural dos slices (figura 3.7), atingindo o nível das células da biblioteca SCLIB.

Com a descrição estrutural dos blocos, tem-se a descrição estrutural do chip até o nível de biblioteca. Para a simulação (validação) estrutural do chip, pode-se incluir a descrição estrutural de apenas um bloco e as descrições comportamentais dos demais para a validação

individual de cada bloco e do próprio chip. Isto possibilita a identificação de erros que possam ocorrer no aprofundamento hierárquico de cada bloco, pois se todos os blocos forem instanciados estruturalmente ao mesmo tempo, fica difícil identificar os blocos que estão ocasionando o erro. Assim, com a instanciação estrutural de apenas um bloco de cada vez, ocorrendo um erro, ele estará neste bloco. este procedimento é adotado para todos os blocos integrantes do chip. Tal metodologia também possibilita a distribuição do trabalho entre diversas equipes de projeto, que poderão otimizar cada bloco componente do sistema, cuja integração se dará à medida que cada bloco for totalmente validado. A consequência imediata já pode ser obtida na redução substancial do tempo de projeto e da possibilidade de fornecer novas versões funcionalmente compatíveis com a especificação, mas com implementações otimizadas setorialmente. Por exemplo, no caso da UAB, qualquer implementação de Somador de 32 bits, poderá substituir a que foi implementada para atender os requisitos de velocidade e área de silício.

Para informar ao simulador ASIMUT que na descrição estrutural do chip existem blocos descritos estruturalmente e outros comportamentalmente, utiliza-se os arquivos de catálogo para determinar as hierarquias do circuito [MAS93o].

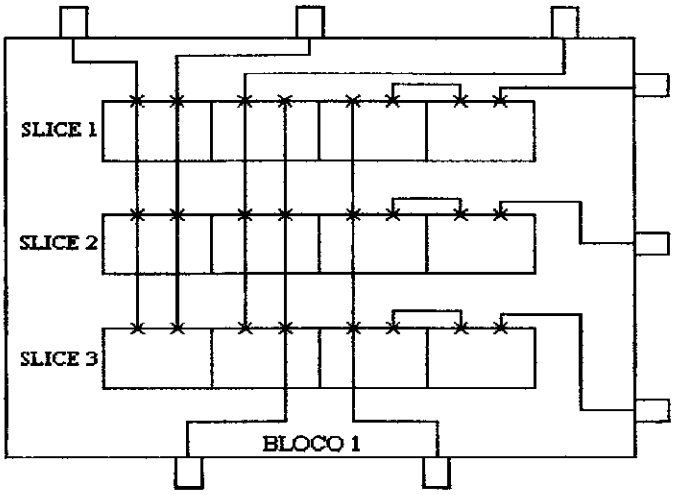


Figura 3.6 - Divisão do Bloco 1 em Slices

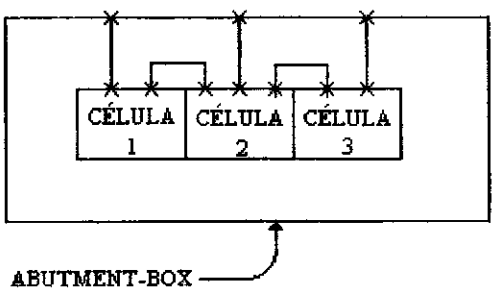


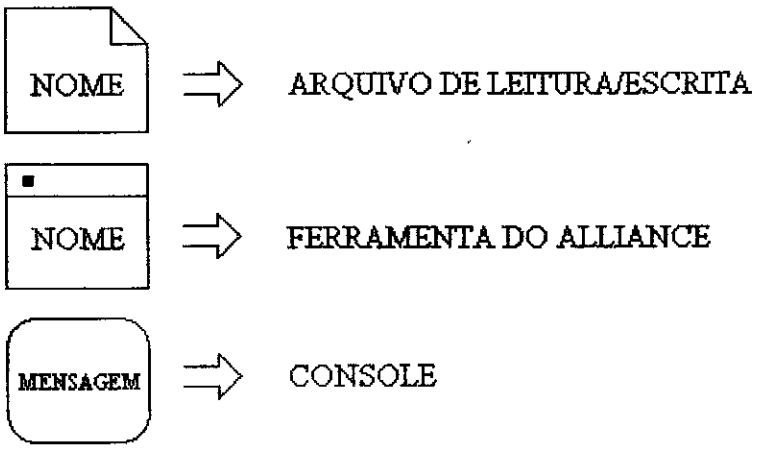
Figura 3.7 - Composição de um Slice com Células de Biblioteca

Observa-se que o objetivo é atingir a descrição estrutural do chip a nível de células de biblioteca. Isto poderia se realizar na etapa 1, mas imagine-se um chip contendo 1000 células. A interligação (NETLIST) destas células individualmente demanda muito tempo, a probabilidade de se fazer uma conexão errada é grande e a depuração de um arquivo de tal tamanho é muito difícil. Portanto, esta metodologia hierarquizada permite só trabalhar com células nos slices e estes, por sua vez, manipulando, em geral um bit operativo. Como o slice é repetitivo dentro do bloco, com o posicionamento instanciado deste, tem-se de imediato toda a NETLIST do bloco, e com as NETLISTS dos blocos tem-se a do chip. Como o chip foi totalmente validado na etapa 1, os blocos na etapa 2, a probabilidade da ocorrência de erro é mínima. Estes possíveis erros serão tratados na seção 5.4.

3.4 - Utilização das Ferramentas do ALLIANCE

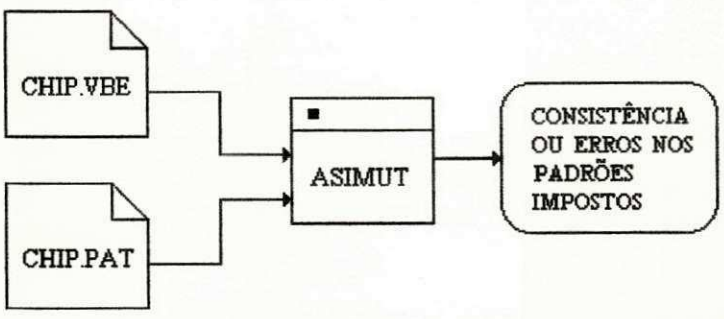
Com base no projeto "TOP-DOWN" hierarquizado com o uso da tecnologia "STANDARD-CELLS", será apresentado a sequência de utilização das ferramentas do ALLIANCE.

Legenda:

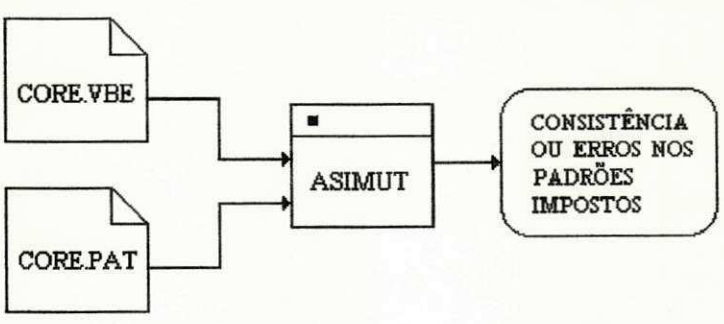


Projeto Lógico

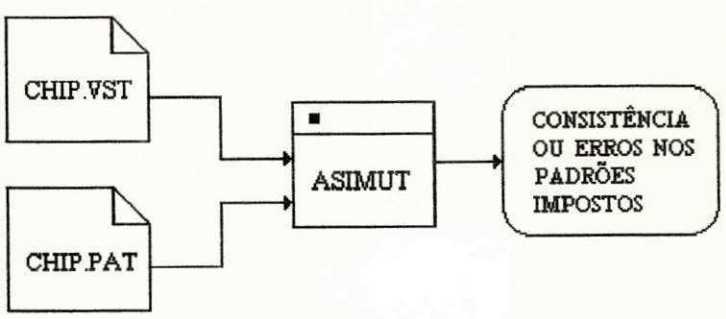
De posse da descrição comportamental do chip (chip.vbe) e dos vetores de teste (chip.pat), pode-se fazer a simulação lógica do circuito com o ASIMUT:



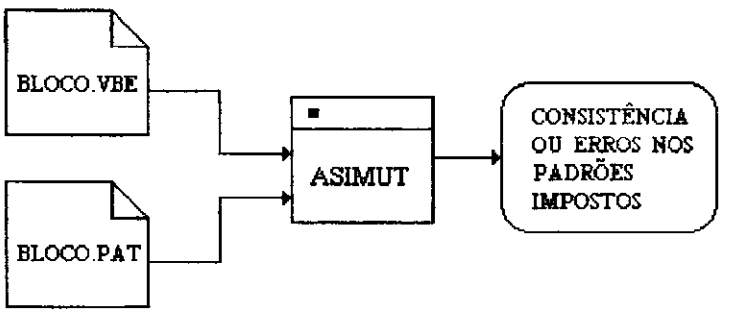
Com a validação do chip na etapa anterior, pode-se validar comportamentalmente o coração (core.vbe) com os vetores de teste escolhidos (core.pat) através do ASIMUT:



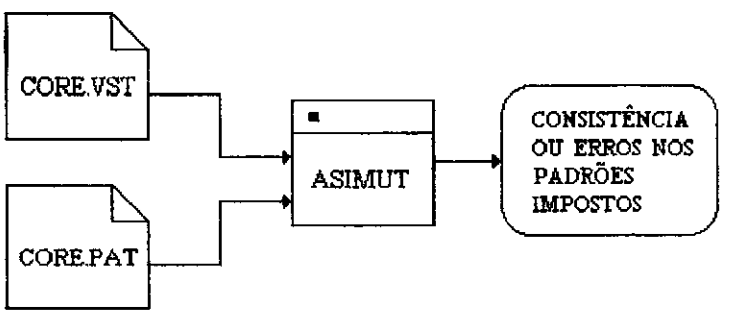
Com a validação do core.vbe, pode-se passar para a descrição estrutural do chip (chip.vst). Nesta descrição encontram-se instanciados o coração (core.vbe) e os "PADS". Os vetores de teste utilizados são os mesmos da validação do chip.vbe:



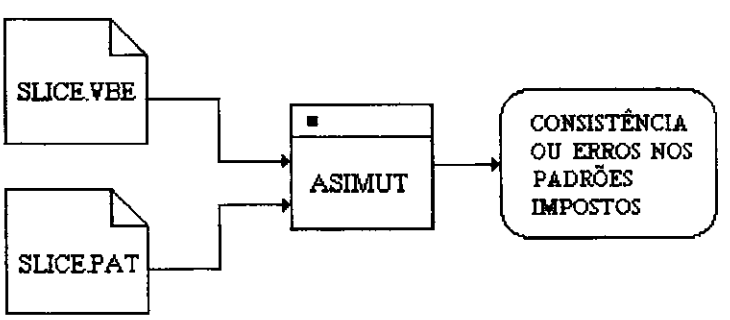
Nesta etapa todos os blocos do coração (bloco.vbe) são validados com seus respectivos vetores de teste (bloco.pat) através do ASIMUT:



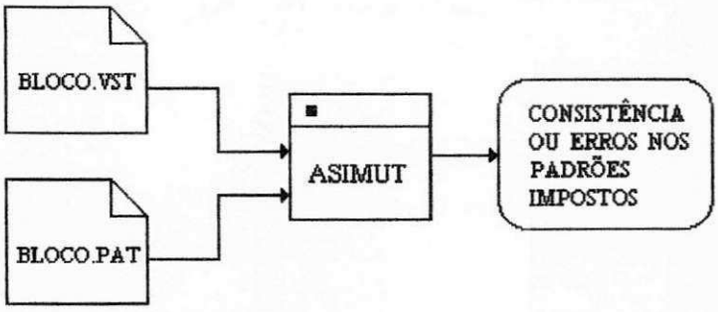
Com os blocos validados comportamentalmente, pode-se validar o estrutural do coração (core.vst). Nesta descrição estrutural, encontram-se instanciados todos os blocos (bloco.vbe) nela contida. Os vetores de teste utilizados são os mesmos da validação do core.vbe:



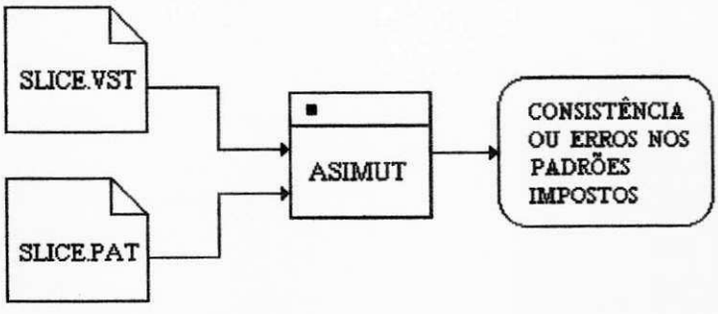
Nesta etapa, todos os slices dos blocos são validados comportamentalmente (slice.vbe) com seus respectivos vetores de teste (slice.pat) através do ASIMUT:



Com os slices validados comportamentalmente, pode-se validar os estruturais dos blocos (bloco.vst) onde se encontram instanciados todos os slices (slice.vbe) do bloco. Os vetores de teste utilizados são os mesmos da validação do bloco.vbe:



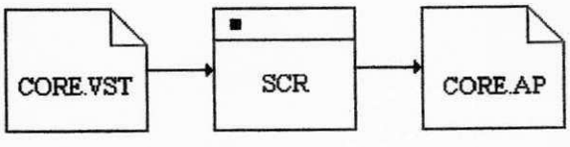
Com os slices validados comportamentalmente, pode-se validar os estruturais dos slices (slice.vst) onde encontram-se instanciadas as células de biblioteca, cujas descrições vbe já estão validadas por definição. Os vetores de teste utilizados são os mesmos da validação do slice.vbe:



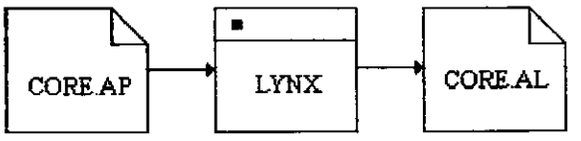
Com isso, tem-se todo o projeto lógico do chip validado.

Projeto Físico

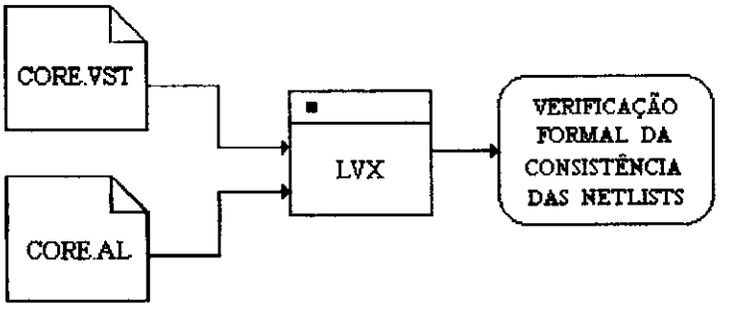
De posse do estrutural do coração (core.vst), pode-se roteá-lo com o SCR, gerando-se assim o core.ap:



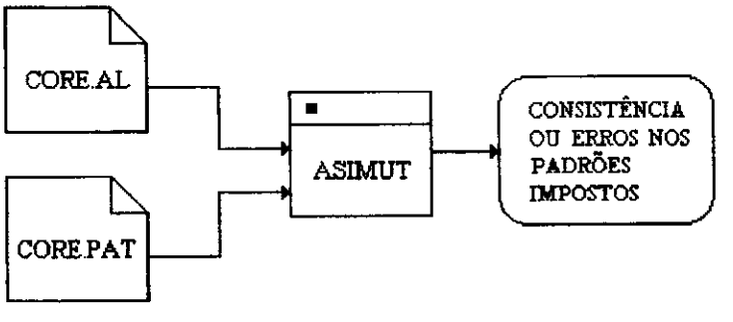
Com o arquivo de "LAYOUT" simbólico, pode-se extrair a "NETLIST" lógica do coração (core.al) com o LYNX:



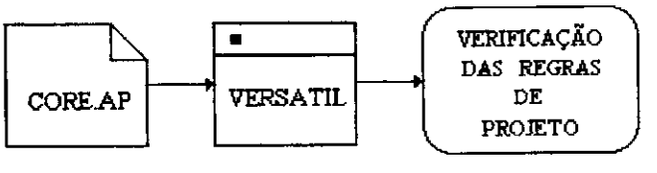
Com a "NETLIST" lógica extraída e com o estrutural do coração (core.vst), pode-se verificar através do LVX, se as duas "NETLISTS" são iguais. Isto é feito para se provar que o "LAYOUT" simbólico gerado, corresponde ao estrutural do coração.



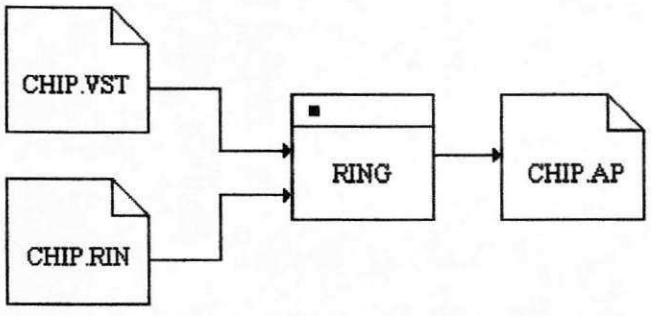
Com a confirmação da etapa anterior, pode-se mais uma vez comprovar através de uma simulação da "NETLIST" lógica (core.al) com os vetores de teste do coração (core.pat):



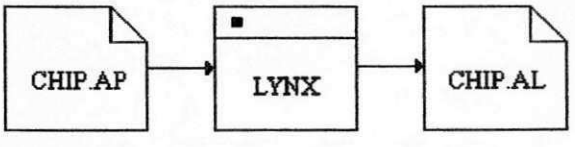
Agora pode-se verificar se na geração do "LAYOUT" foram desrespeitadas regras de projeto. Isto é feito através do VERSATIL:



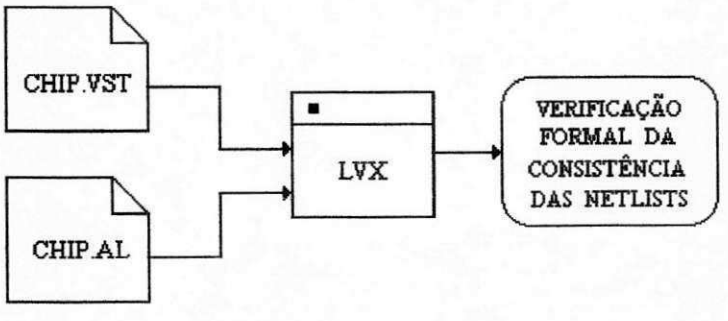
Com a aprovação da etapa anterior, pode-se agora rotear o coração com a coroa de "PADS". Para isso, utiliza-se o arquivo chip.rin para informar ao RING a posição dos "PADS". Este arquivo informa o lado (norte, sul, leste ou oeste) e o local onde este será colocado.



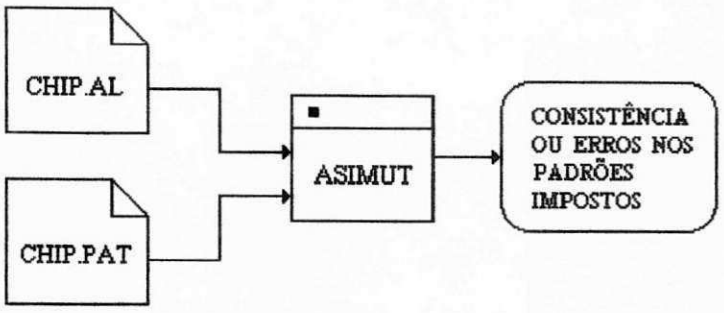
Com o "LAYOUT" simbólico do chip, pode-se extrair a "NETLIST" lógica do mesmo, através do LYNX:



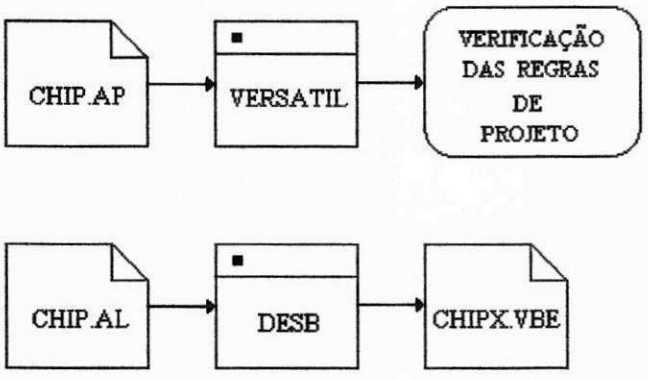
Com a "NETLIST" lógica extraída e com o estrutural do chip (chip.vst), pode-se verificar através do LVX, se as duas "NETLISTS" são iguais. Isto é feito para se provar que o "LAYOUT" simbólico representa o estrutural do chip (coração mais "PADS").



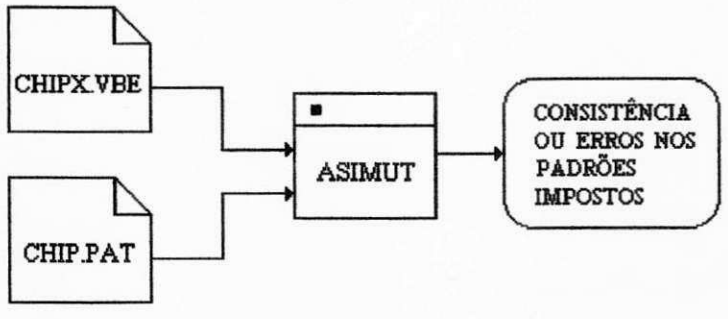
Com a confirmação da etapa anterior, pode-se mais uma vez comprovar através de uma simulação da "NETLIST" lógica (chip.al) com os vetores de teste do chip (chip.pat);

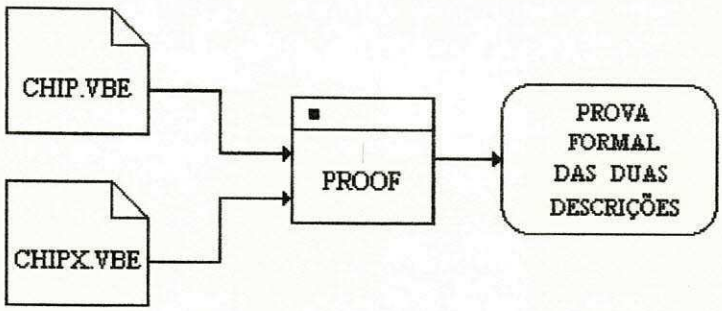


Agora pode-se verificar se não foi desrespeitada nenhuma regra de projeto. Isto é feito através do VERSATIL. Também é feita a extração, a partir da "NETLIST" lógica, da descrição comportamental do chip (chipx.vbe):

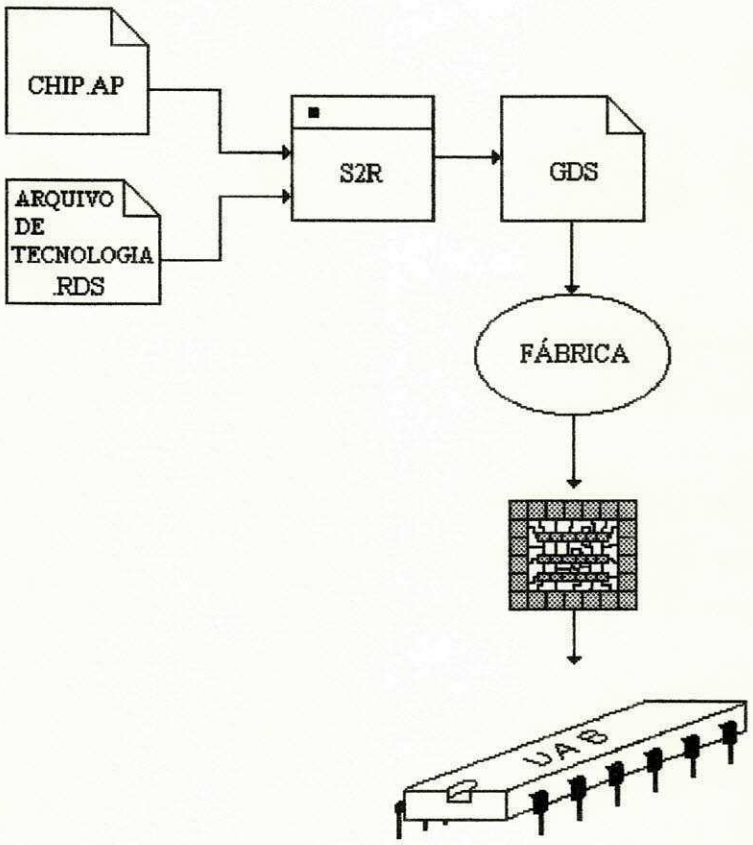


Para verificar-se que a descrição comportamental extraída (chipx.vbe) é a mesma desenvolvida, faz-se a simulação lógica do chipx.vbe com o chip.pat através do ASIMUT, e a prova formal com o PROOF:





Com o arquivo de "LAYOUT" simbólico e com o arquivo de tecnologia, pode-se gerar o formato de "LAYOUT" real através do S2R:



Deve-se também, dentre os vetores de teste que serviram à validação funcional, selecionar um conjunto o mais reduzido possível, que servirá ao teste do chip logo após sua produção, ainda no "wafer". Este deverá ser reduzido pois o tempo de simulação feita na usina, custará extremamente caro. Deverá ser o mais exaustivo possível quanto à detecção de falhas, pois se passar ao encapsulamento com estas, os custos estarão sendo multiplicado por 10.

CAPÍTULO IV

METODOLOGIA DO PROJETO

A Unidade Aritmética Básica (UAB) realiza operações de Adição, Subtração, Multiplicação e Divisão. Assim, para a implementação da UAB, Precisa-se determinar que tipos de algoritmos serão utilizados. De posse desses algoritmos, verificam-se as partes em comum entre eles, de forma a se desenvolver uma unidade de controle que faça as comutações entre estas partes, de tal forma que um único circuito realize as quatro operações. Esta foi a metodologia adotada no projeto da UAB.

Este capítulo apresenta uma ligeira discussão sobre algoritmos paralelos e sequenciais, de forma a se justificar o tipo escolhido e a implementação individual, de cada operação, em "VHDL".

4.1 - Algoritmos Paralelos e Sequenciais

A metodologia adotada no projeto da UAB é a de implementar o Somador, o Subtrator, o Multiplicador e o Divisor em "VHDL". De posse dos quatro circuitos, reúne-se em um único circuito a mesclagem destes, assim, os algoritmos escolhidos devem propiciar este tipo de metodologia.

Os algoritmos paralelos apresentam como principal vantagem a velocidade e como desvantagem, eles exigem uma área de silício muito grande. Estes algoritmos geralmente utilizam uma complexa lógica combinacional, para realizar uma determinada operação, por isso são sistemas que não permitem compartilhar com outros, partes do seu circuito, ou seja, são sistemas dedicados. Assim, para se desenvolver um circuito como o proposto, seria necessário uma arquitetura como a mostrada na figura 4.1.

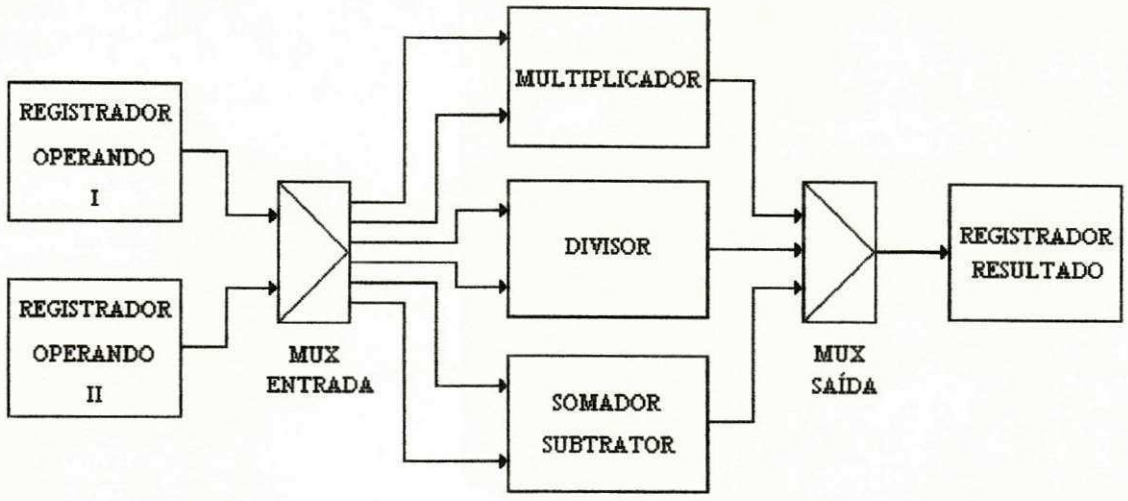


Figura 4.1 - Unidade Aritmética Utilizando Algoritmos Paralelos

Observa-se que neste circuito existem internamente três subcircuitos complexos, que ocupam uma grande área de silício cada um e a área de todo o circuito será muito grande, o que pode inviabilizar a confecção de um ASIC.

Já os algoritmos sequenciais são compostos por uma pequena lógica combinacional e uma grande parte de registradores. Estes, por sua vez, permitem a manipulação dos seus valores e seu armazenamento; assim, é possível compartilhar um registrador com dois ou mais circuitos. Com isso, a área necessária para a realização de um circuito como a UAB, torna-se bem menor, pois a maior parte da área dos circuitos envolvidos são reduzidas a alguns registradores comuns em todos eles.

A principal vantagem dos circuitos sequenciais é a área de silício necessária à sua implementação, ser bem inferior a dos paralelos e sua principal desvantagem é a velocidade que é bem menor que a dos paralelos. Para compatibilizar estes dois aspectos, área e velocidade, observa-se de início, que os algoritmos sequenciais de Multiplicação e Divisão tem como base um Somador/Subtrator. Este por sua vez, se for do tipo sequencial, demandará muito tempo para a realização de cada operação. Por isso foi estudada uma configuração com soma/subtração do tipo paralelo e Multiplicação/Divisão sequenciais. A área do Somador/Subtrator paralelo é bem menor que a de um Multiplicador ou Divisor, e a velocidade final desta associação é muito boa (ver seção 4.2). A arquitetura da UAB usando esta associação, mostrada na figura 4.2, foi a escolhida no presente trabalho.

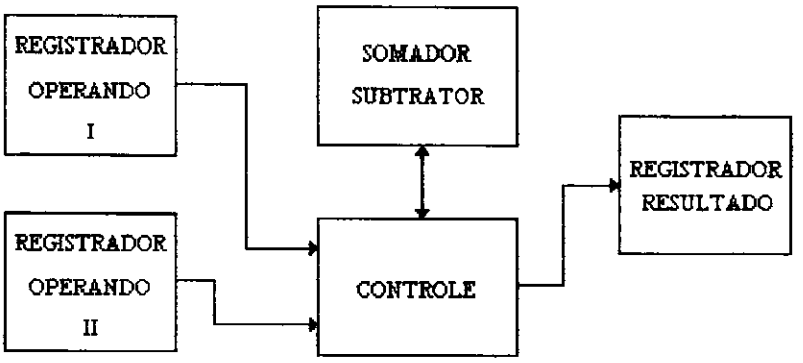


Figura 4.2 - Unidade Aritmética Utilizando Algoritmos Sequenciais

A seguir serão apresentados os algoritmos adotados nas implementações do Somador/Subtrator, Multiplicador e Divisor.

4.2 - Somador/Subtrator Paralelo

Esta seção apresenta o Somador Carry Lookahead de 32 bits, utilizado no projeto da UAB. Também será apresentado o módulo de complemento a dois, para as realizações das operações de subtração e divisão.

4.2.1 - Somador Carry Lookahead

A técnica Carry Lookahead ("vai-um" antecipado) [HWA79], é uma das mais rápidas implementações de somadores, guardando pequena complexidade de hardware, por isso foi o algoritmo adotado no projeto da UAB.

Como todo Somador, o resultado é obtido da soma de duas parcelas com o "carry" anterior:

$$S_i = A_i \oplus B_i \oplus C_{i-1} \qquad (4.1)$$

Assim, a velocidade depende de como são gerados estes "carry's" anteriores. Neste algoritmo eles não são propagados, eles são calculados por um "Bloco Carry Lookahead", daí o nome do algoritmo. A seguir serão mostrados os cálculos realizados por este bloco.

Fazendo:

$$P_i = A_i \oplus B_i \qquad (4.2)$$

tem-se em (4.1)

$$S_i = P_i \oplus C_{i-1} \qquad (4.3)$$

mas

$$C_i = A_i . B_i + S_i \qquad (4.4)$$

fazendo:

$$G_i = A_i . B_i \qquad (4.5)$$

tem-se em (4.4)

$$C_i = G_i + P_i . C_{i-1} \qquad (4.6)$$

Assim,

$$C_0 = G_0 + P_0 C_{IN}$$

$$C_1 = G_1 + P_1 C_0 = G_1 + P_1 G_0 + P_0 P_1 C_{IN}$$

$$C_2 = G_2 + P_2 C_1 = G_2 + P_2 G_1 + P_1 P_2 G_0 + P_0 P_1 P_2 C_{IN}$$

$$\vdots$$

$$C_{N-1} = G_{N-1} + P_{N-1} G_{N-2} + \cdots + P_0 P_1 \cdots P_{N-1} C_{IN} \qquad (4.7)$$

Os circuitos que satisfazem estas equações são mostrados a seguir:

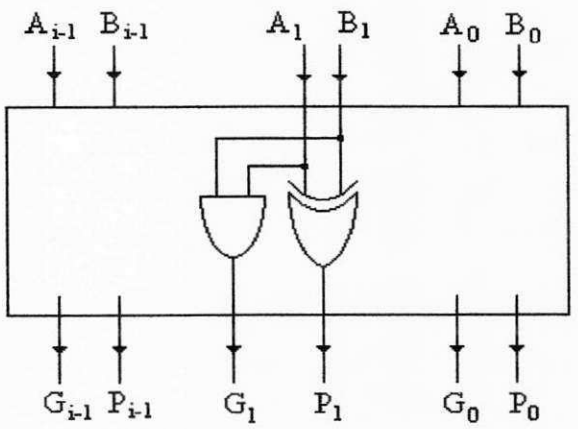


Figura 4.3 - Unidade de Propagate (P_i) e Generate (G_i)

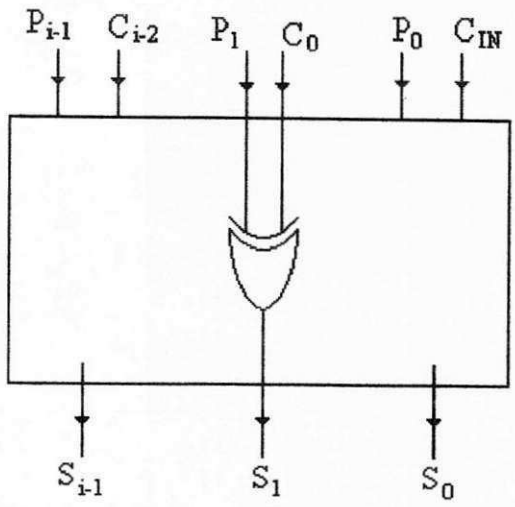


Figura 4.4 - Unidade de Soma

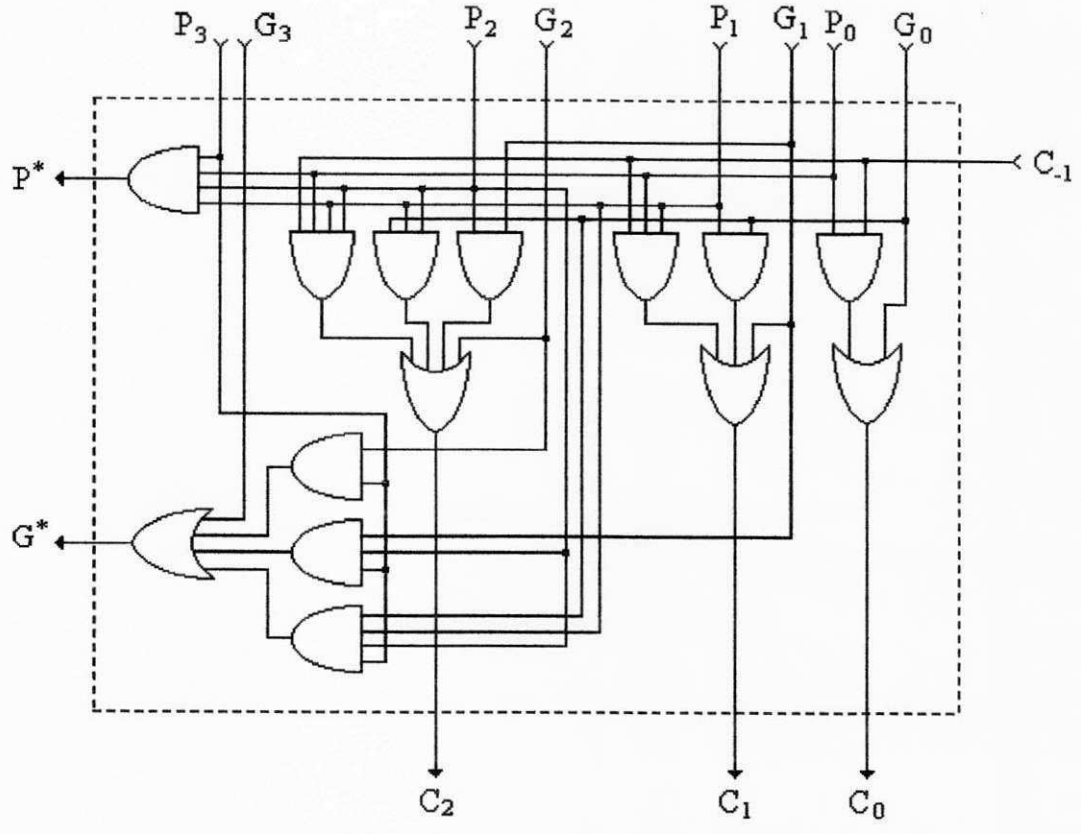


Figura 4.5 - Bloco de Carry Lookahead de 4 Bits

$$P^* = P_0 P_1 P_2 P_3 \qquad (4.8)$$

$$G^* = G_3 + G_2 P_3 + G_1 P_2 P_3 + G_0 P_1 P_2 P_3 \qquad (4.9)$$

O diagrama de blocos do Somador Carry Lookahead de 32 bits, é mostrado na figura 4.6.

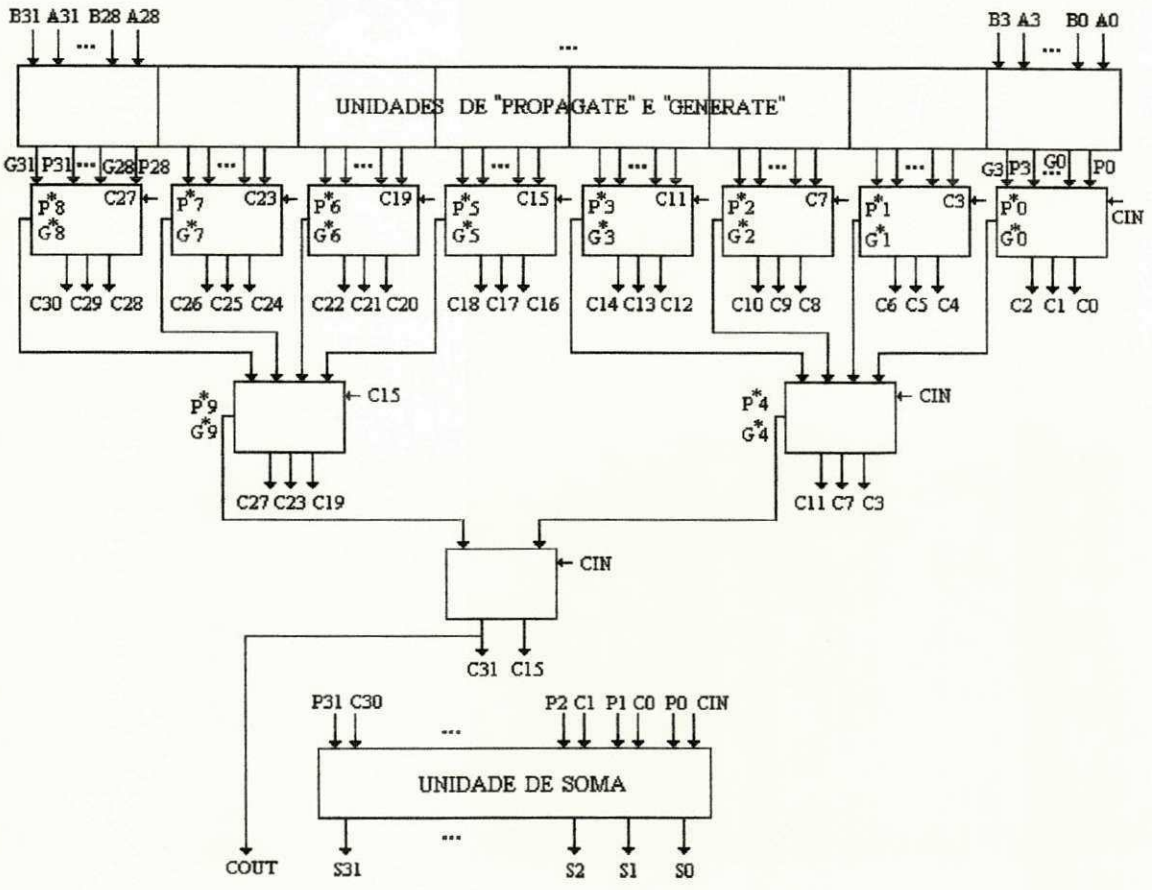


Figura 4.6 - Somador de 32 Bits

A descrição "VHDL" deste Somador está mostrada no ANEXO I.

Observa-se, pela equação 4.7 e pela figura 4.5, que

$$C_0 = G_0 + P_O C_{IN}$$
$$C_1 = G_1 + P_1 G_0 + P_0 P_1 C_{IN}$$
$$C_2 = G_2 + P_2 G_1 + P_1 P_2 G_0 + P_0 P_1 P_2 C_{IN}$$

A simples passagem destas equações para o hardware não constitui a melhor solução, pois através de uma pesquisa na biblioteca SCLIB, verifica-se que os tempos de atrasos destes sinais são 5.26 ns, 6.35 s, 8.97 ns, respectivamente. Assim, um Somador de 32 bits

implementado com estas equações terá um atraso total de 46.14 ns, ou seja, ele pode operar no máximo a 21.7 MHz.

Aplicando-se o teorema de DE MORGAN nestas equações, tem-se:

$$C_0 = \overline{\overline{G_0} \cdot \overline{P_0 C_{IN}}}$$

$$C_1 = \overline{\overline{G_1} \cdot \overline{P_1 G_0} \cdot \overline{P_0 P_1 C_{IN}}}$$

$$C_2 = \overline{\overline{G_2} \cdot \overline{P_2 G_1} \cdot \overline{P_1 P_2 G_0} \cdot \overline{P_0 P_1 P_2 C_{IN}}}$$

Com isso, os tempos de atrasos se reduzirão à 3.64 ns, 4.22 ns e 5.72 ns, levando a um atraso total de 31.56 ns, no projeto de um Somador de 32 bits. Isto possibilita uma operação a 31.7 MHz. A figura 4.7 mostra a implementação, em hardware, destas equações. Este foi o hardware adotado no circuito da figura 4.5.

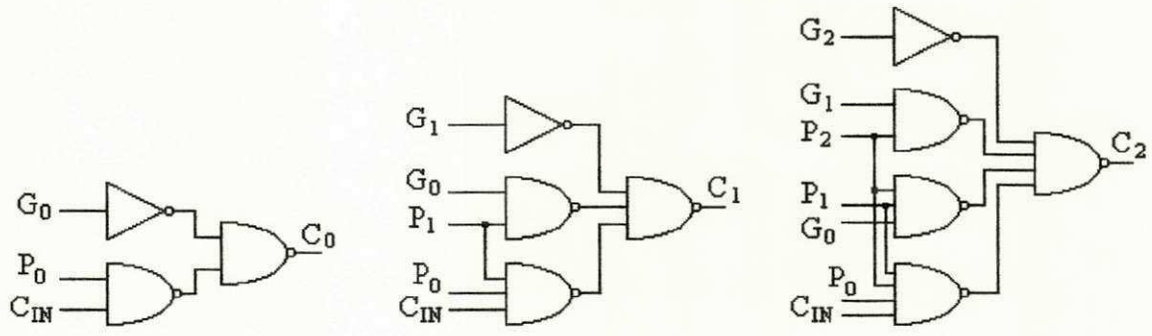


Figura 4.7 - Hardware Adotado no Bloco Carry Lookahead

4.2.2 - Subtrator

Para a realização da Subtração, faz-se o complemento a dois da parcela negativa e soma-se com a outra. Para se fazer o complemento a dois, basta complementar o respectivo número e somar com '1'. O complemento é feito com inversores e a soma com '1' é feita com o "carry" de entrada C_{in}, pois:

$$S = A + B + C_{IN}$$

Assim, para implementação de um circuito Somador/Subtrator, basta colocar um bloco que realize o complemento e faça C_{in} = 1 quando for subtrair, ou o contrário para somar. Este circuito está mostrado na figura 4.8.

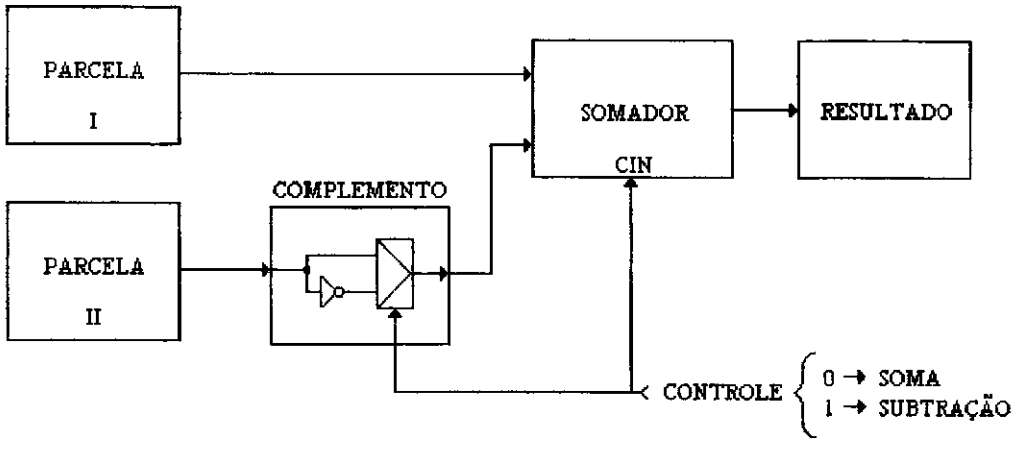


Figura 4.8 - Somador/Subtrator

4.3 - Multiplicador

O algoritmo de multiplicação sequencial adotado no projeto da UAB, foi o "Lápis e Papel", por ser simples e bem rápido. Conforme citado no capítulo 1, este Multiplicador será de 16 bits, fornecendo resultados em 32 bits.

Para explicar o algoritmo "Lápis e Papel" da multiplicação, faz-se necessário a observação de uma multiplicação binária. Sejam dois números A=1100110 (multiplicando) e B=10101 (multiplicador), assim AxB será:

					1	1	0	0	1	1	0	multiplicando
							1	0	1	0	1	multiplicador
					1	1	0	0	1	1	0	L1
				0	0	0	0	0	0	0		L2
			1	1	0	0	1	1	0			L3
		0	0	0	0	0	0	0				L4
	1	1	0	0	1	1	0					L5
1	0	0	0	0	1	0	1	1	1	1	0	produto

Observa-se, pelo cálculo, que quando um bit do multiplicador é "1", sua respectiva parcela é igual ao multiplicando, caso contrário é "0". O produto é obtido com a soma deslocada de cada parcela.

Passando a analisar a operação, do ponto de vista do circuito, observa-se que em nenhum momento o multiplicando sofre alguma alteração, assim ele pode ser visto como um registrador de carregamento paralelo. Já o multiplicador é tratado por bit, sempre no sentido da direita para a esquerda, o que dá a idéia de um registrador de deslocamento á direita, onde o bit a ser lido é o menos significativo. Quando este bit é "1", deixa-se passar o multiplicando, caso contrário, passa "0", assim precisa-se de um controle na saída do registrador do multiplicando. Este controle é realizado por uma porta AND em cada bit do multiplicando com o bit menos significativo do multiplicador. Assim o controle fará a operação desejada.

Para a obtenção do produto, observa-se pelas linhas de 1 a 5 (L1 a L5), que trata-se de um Somador cujo número de bits é o mesmo do Multiplicador e que a cada soma o produto é deslocado à direita. Assim, o produto é armazenado em um registrador de deslocamento à direita. Como esta soma pode ser realizada passo a passo, isso induz um armazenamento intermediário no registrador de produto, pois ele soma e depois desloca, assim, além de deslocar à direita, este registrador permite um carregamento paralelo.

Com essas observações, pode-se dizer que o Somador recebe um valor resultado do controle do multiplicador sobre o multiplicando e outro da própria saída, ou seja, do produto.

De posse destes conhecimentos, pode-se definir um projeto de um Multiplicador "Lápis e Papel", para isso, define-se a interface do circuito (figura 4.9). O diagrama de tempo deste Multiplicador está mostrado na figura 4.10.

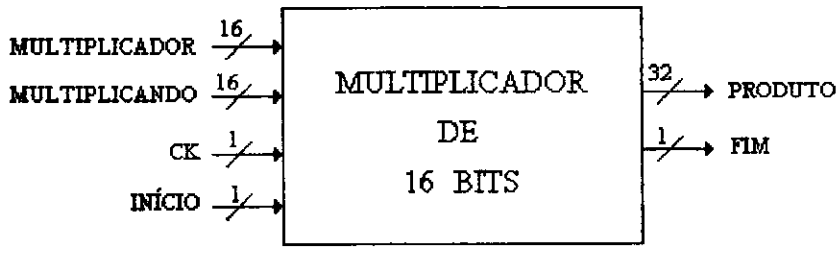


Figura 4.9 - Interface do Multiplicador "Lápis e Papel"

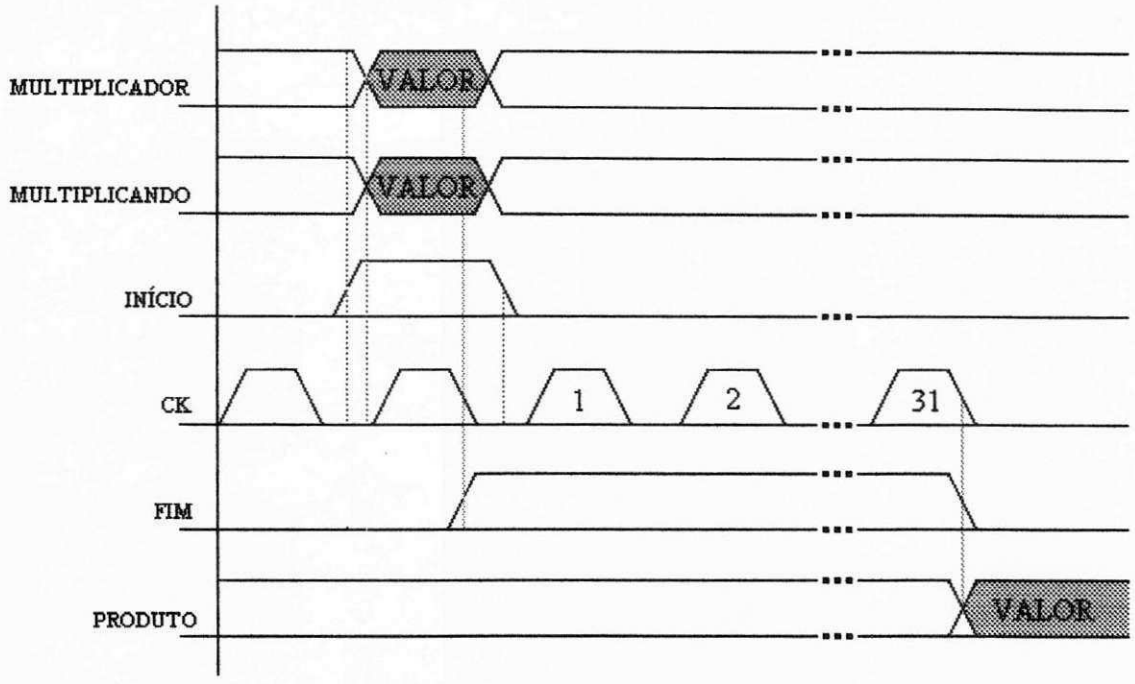


Figura 4.10 - Diagrama de Tempo do Multiplicador

Quando o sinal de INÍCIO vai a "1", coloca-se nas respectivas entradas, os valores de multiplicando e multiplicador. Quando início cai a "0", dá-se INÍCIO a multiplicação, o sinal do FIM vai a "1", indicando que uma operação está em desenvolvimento. Quando FIM cai a "0", indica que a multiplicação terminou e o resultado está disponível no registrador produto.

A figura 4.11 mostra o circuito completo do Multiplicador. O Somador utilizado foi desenvolvido na seção 4.2, só que de 16 bits (como pode-se ver pelo exemplo tomado para explicar o algoritmo "Lápis e Papel", as somas são realizadas com números de 16 bits). A descrição "VHDL" do Multiplicador está no ANEXO II. Para um clock de 33 MHz (utilizando-se os dados da biblioteca SCLIB), uma multiplicação será realizada em 0.94 μs.

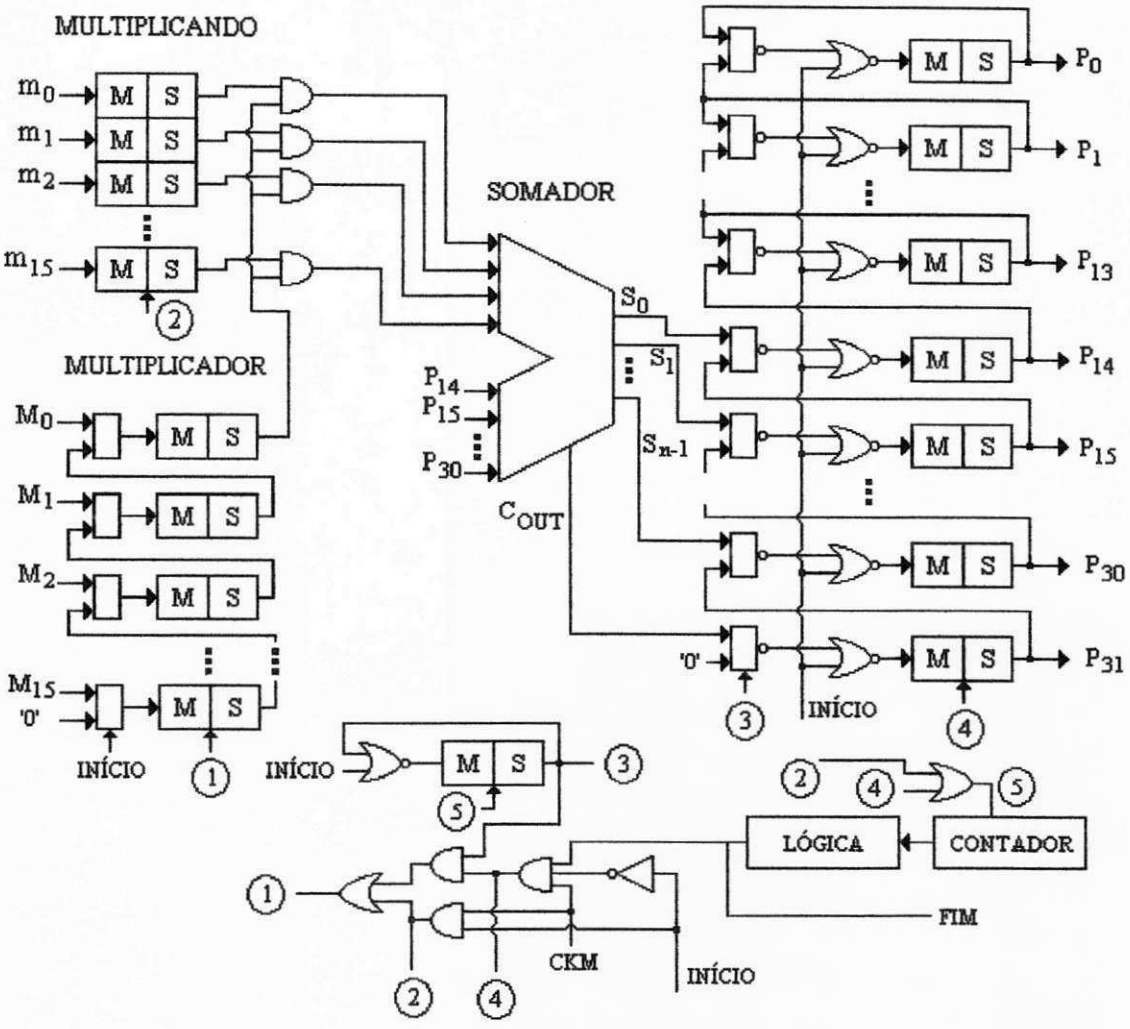


Figura 4.11 - Multiplicador "Lápis e Papel" de 16 Bits

A lógica combinacional da figura 4.11, consta da unidade de controle do Multiplicador. O contador utilizado é necessário para a contagem dos pulsos de clock para a realização da operação. No caso dos 16 bits, o número de pulsos é de 31 (2N-1, onde N é o número de bits do Multiplicador). Assim, quando o contador atingir 31 (11111B), uma lógica bloqueia o clock, evitando que outras sequências sejam desencadeadas, e o sinal FIM cai a '0'. O contador e a lógica serão mostrados na próxima seção.

4.3.1 - Contador e Lógica de Controle

O projeto de um contador em "VLSI", é um pouco diferente do projeto discreto, porque na versão utilizada da "SCLIB" não existe uma célula de um flip-flop com Q e $\overline{Q}$, mas

sim latchs. Assim, é necessário uma lógica para gerar as sequências de uma contagem binária. A figura 4.12 mostra o diagrama de blocos de um contador de três bits e a tabela 4.1 mostra a tabela da verdade.

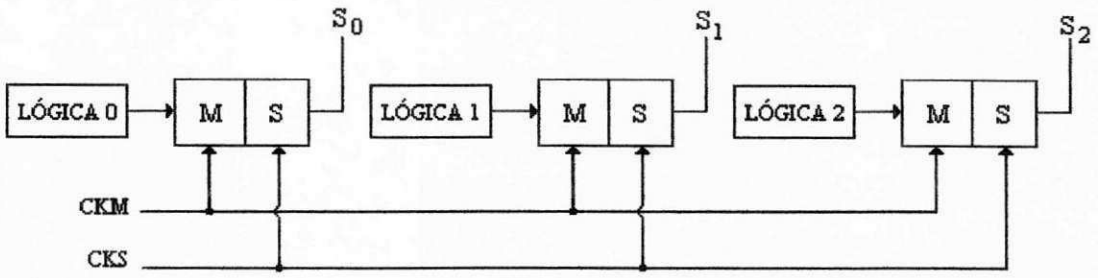


Figura 4.12 - Diagrama de Blocos de um Contador de Três Bits

S_2	S_1	S_0	L_2	L_1	L_0
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	0

Tabela 4.1 - Tabela da Verdade de um Contador de Três Bits

Pela tabela tira-se:

$L_0 = \overline{S_0}$

$L_1 = S_0 \oplus S_1$

$L_2 = S_2 \oplus (S_1 \cdot S_0)$

Na seção anterior, viu-se que para um Multiplicador de 16 bits, a contagem deveria ser até 31. Assim, este contador deverá ter 5 bits. Por expansão das expressões acima, temos:

$L_3 = S_3 \oplus (S_2 \cdot S_1 \cdot S_0)$

$L_4 = S_4 \oplus (S_3 \cdot S_2 \cdot S_1 \cdot S_0)$

O circuito do contador de 5 bits com reset está mostrado na figura 4.13.

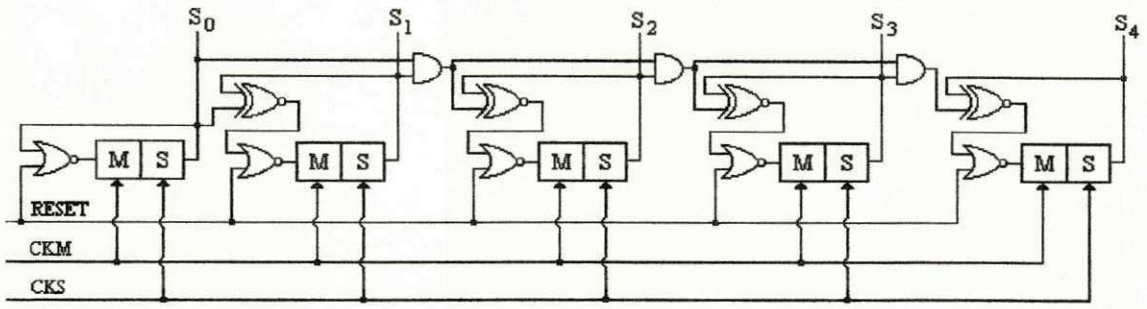


Figura 4.13 - Contador de 5 Bits com Reset

A lógica que leva o sinal FIM a "0" e trava o clock, deve gerar um "0" na saída quando o contador atingir 31, ou em binário, "11111", assim:

$$\text{Lógica} = \overline{S_0 \cdot S_1 \cdot S_2 \cdot S_3 \cdot S_4}$$

Como na SCLIB, o maior número de entradas encontrado é quatro, a lógica será implementada segundo a figura 4.14.

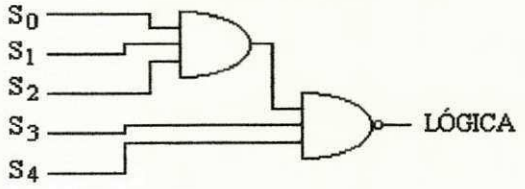


Figura 4.14 - Circuito da Lógica de Controle

4.4 - Divisor

O Divisor implementado na UAB utiliza o algoritmo "Lápis e Papel" devido sua simplicidade de hardware e bom desempenho (velocidade). O Divisor será de 32 bits.

Para apresentação deste algoritmo, fará-se-á a divisão binária de dois números inteiros positivos. Seja A=10110011 (dividendo) e B=1010 (divisor), o quociente será obtido da seguinte maneira:

	Dividendo								Divisor				
	1	0	1	1	0	0	1	1	1	0	1	0	
-	1	0	1	0					1	0	0	0	1 Quociente
	0	0	0	1									
		0	0	1	0								
Restos			0	1	0	0							
Parciais				1	0	0	1						
					1	0	0	1	1				
					-	1	0	1	0				
			Resto		0	1	0	0	1				

Observa-se que esta divisão se desenvolve da mesma forma que a decimal, ou seja, quando um resto parcial é maior que o divisor, coloca-se "1" no quociente e baixa-se o próximo dígito do dividendo ao resto parcial. Caso contrário, coloca-se "0" no quociente e procede-se da mesma forma. Este fato leva a uma relação entre o resto parcial com o divisor e o dígito a se colocar no quociente. Como a análise foi feita no caso do resto parcial ser maior ou não que o divisor, fará-se-á uma verificação. No início da divisão tem-se:

Resto Parcial = 1011

Como o divisor é sempre o subtrator, faz-se o complemento a dois deste sem alterar o número de bits dele. Assim, tem-se:

Divisor = 1010 => Divisor Negativo = 0110

assim,

	1	0	1	1
+	0	1	1	0
1	0	0	0	1

Observa-se que o bit em destaque é o carryout. Seguindo-se com a divisão, tem-se:

Resto Parcial = 0010

Assim,

	0	0	1	0
+	0	1	1	0
0	1	0	0	0

Analisando os dois casos acima, observa-se que o valor do dígito a ser colocado no quociente é igual ao carryout. Esta é uma característica do algoritmo que faz com que o hardware seja bem simplificado.

Analisando a operação de divisão, do ponto de vista do circuito, observa-se que o divisor não sofre qualquer manipulação, assim ele é um registrador de carregamento paralelo. Uma característica deste registrador é que ele é inversor, pois o divisor é sempre o número subtrator, já que a divisão é realizada com um Subtrator, conforme visto acima. Conforme a descrição do Subtrator na seção 4.2.2, esta parcela é complementada e o carryin igual a "1".

No caso do dividendo, observa-se que os bits que descem para o resto parcial, deslocam-se no sentido da esquerda para a direita, assim ele é um registrador de deslocamento à esquerda. O mesmo ocorre com o quociente.

Uma outra característica do circuito, é o fato do resto parcial ser sempre armazenado, o que implica no adicionamento de mais um registrador para guardar o valor do resto. Os bits que entram neste registro, vêem do dividendo e de forma serial à esquerda, ou do subtrator quando o "carryout" é "1".

Com isso, definem-se de forma básica, os elementos integrantes do Divisor. Assim, pode-se definir uma interface deste circuito, é o que mostra a figura 4.15.

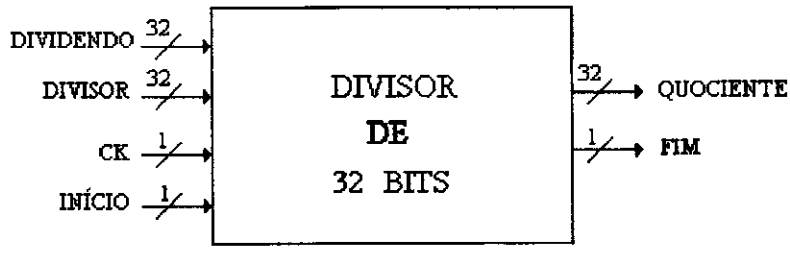


Figura 4.15 - Interface do Divisor "Lápis e Papel"

O diagrama de tempo deste circuito é exatamente igual ao do Multiplicador, mostrado na figura 4.10.

O circuito completo do Divisor binário de 32 bits é mostrado na figura 4.16. Sua descrição em "VHDL" está no ANEXO III.

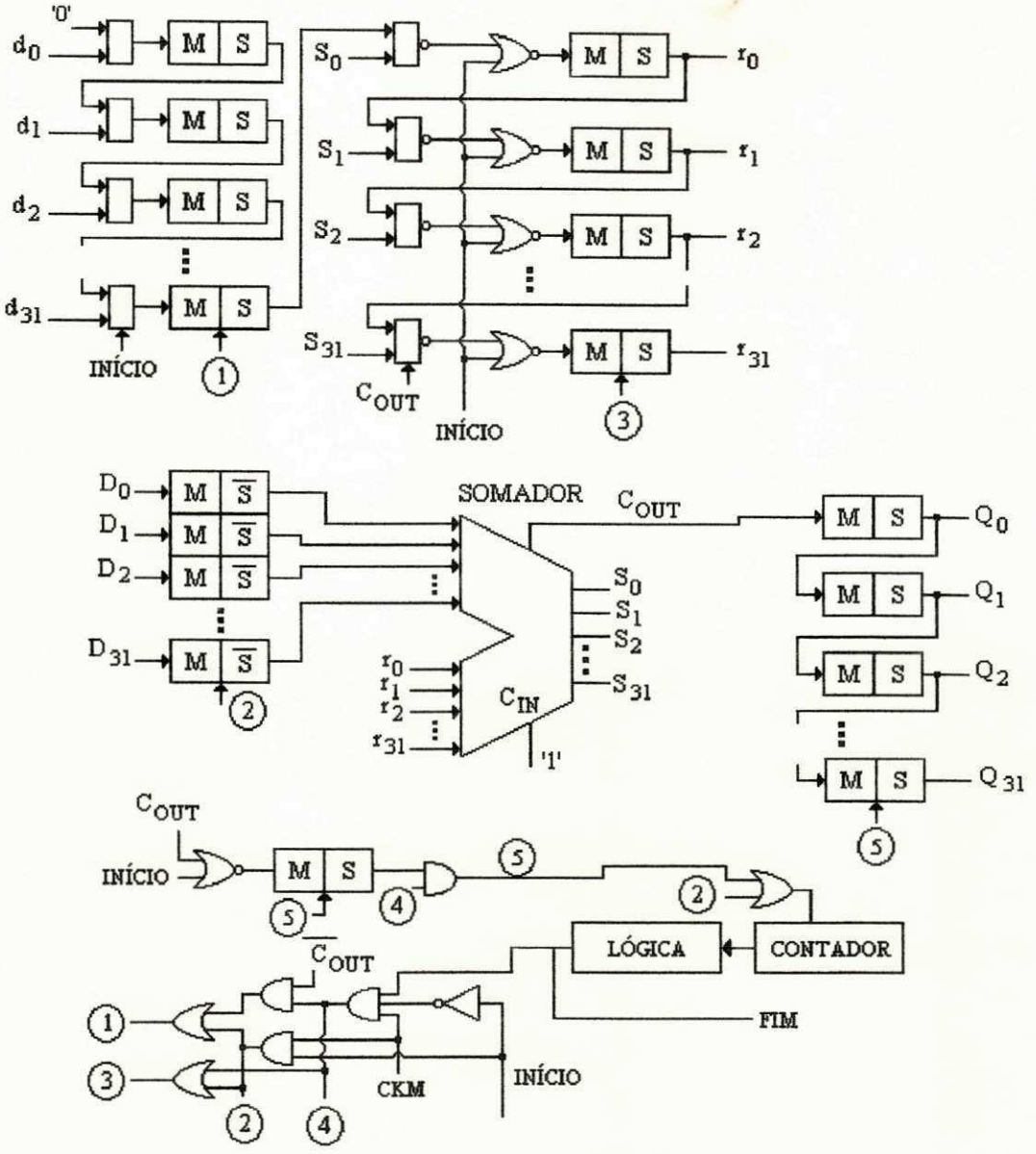


Figura 4.16 - Divisor "Lápis e Papel" de 32 Bits

O Somador utilizado foi o mesmo mostrado na seção 4.2. O contador utilizado é de 6 bits. A contagem indicando o final da divisão é 32, pois neste circuito, os ciclos citados são os pulsos que fazem o deslocamento à esquerda do quociente que é um registrador de 32 bits. A lógica que trava o clock do Divisor e leva o sinal FIM para "0" é um simples inversor no bit 5 do contador, ver figura 4.17.

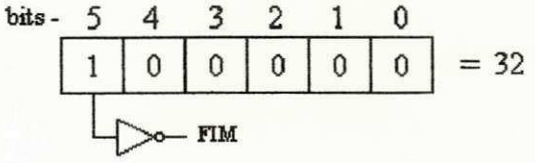


Figura 4.17 - Lógica de Travamento do Divisor

Quanto à velocidade, precisa-se de uma análise para determina-la. Voltando ao exemplo da divisão, no início desta seção, observa-se que quando o "carryout" é "1", o resto parcial armazena o resultado da subtração e executa um deslocamento à esquerda colocando o próximo bit do dividendo no seu bit menos significativo. Isto exige dois ciclos de clock para essa operação. Quando o carryout é "0", o resto parcial sofre apenas um deslocamento à esquerda, semelhante ao caso anterior. Isto exige apenas um ciclo de clock. Assim, conclui-se que a velocidade do divisor depende do número de "0" e "1" que apareçam no quociente. Com esta análise, observa-se que o menor tempo (melhor caso) para a realização de uma divisão é quando o quociente é todo "0". Neste caso, só teremos deslocamentos no resto parcial, implicando num total de 32 ciclos de clock (este caso ocorre quando o dividendo é menor que o divisor). O maior tempo (pior caso) exigido para a divisão é quando o quociente é todo "1", neste caso, cada bit "1" do quociente implica em dois ciclos de clock, levando, portanto, 64 ciclos de clock. Este caso ocorre quando o dividendo é "FFFFFFFF" e o divisor é "00000001". Para um clock de 33 MHz, o tempo de uma divisão (de acordo com os dados da SCLIB) estará entre 0.97 μs e 1.94 μs. Como o pior caso determina a velocidade do sistema, o segundo valor define o tempo necessário à realização de uma divisão.

4.5 - Raiz Quadrada

A Raiz Quadrada é mais uma operação aritmética realizada no Transdutor Digital. Esta seção apresenta dois algoritmos para implementação desta operação. Um utiliza o método da multiplicação e o outro o da divisão. Depois será feita uma comparação entre os dois algoritmos apresentados e a solução adotada no projeto da UAB.

4.5.1 - Raiz Quadrada: Método da Multiplicação

Este método se baseia na técnica de aproximações sucessivas, onde é dado um "chute" inicial para o valor da Raiz. A partir deste valor, os bits são encontrados após as realizações consecutivas de multiplicação e comparação. A operação de multiplicação utilizada é a de quadrado.

Para exemplificar este método, considera-se um número inteiro positivo de 8 bits **A** (radicando). A Raiz de **A** é um número de 4 bits. Assim, o "chute" inicial da Raiz é "1000". Este número é elevado ao quadrado e comparado com **A**. Se **A** for maior ou igual que este "chute", o bit mais significativo dele é "1", caso contrário passará a "0". Com isso, passa-se para o próximo bit, assim o novo "chute" será X100. Pega-se este valor e eleva-se mais uma vez ao quadrado e compara-se com **A**. O valor real deste bit segue o mesmo procedimento adotado no primeiro bit. Segue-se dessa forma até o bit menos significativo da Raiz.

Para ilustrar o algoritmo, considere **A**=10101001 (169D). A raiz de **A** (**RA**) será :

1º chute $\Rightarrow$	**RA**=1000 $\Rightarrow$	**RA**^{2}=01000000 $\Rightarrow$	**A** $\geq$ **RA**^{2} $\Rightarrow$	**RA**(3) = 1
2º chute $\Rightarrow$	**RA**=1100 $\Rightarrow$	**RA**^{2}=10010000 $\Rightarrow$	**A** $\geq$ **RA**^{2} $\Rightarrow$	**RA**(2) = 1
3º chute $\Rightarrow$	**RA**=1110 $\Rightarrow$	**RA**^{2}=11000100 $\Rightarrow$	**A** $<$ **RA**^{2} $\Rightarrow$	**RA**(1) = 0
4º chute $\Rightarrow$	**RA**=1101 $\Rightarrow$	**RA**^{2}=10101001 $\Rightarrow$	**A** $\geq$ **RA**^{2} $\Rightarrow$	**RA**(0) = 1

Assim,

RA = 1101

Se um número possui n bits, sua Raiz Quadrada terá n/2 bits. O número de quadrados e comparações realizadas é igual ao número de bits da Raiz, ou seja, n/2. No caso de uma operação de 32 bits, este número (n/2) será 16. Com isso, pode-se determinar o tempo necessário a realização de uma operação de Raiz Quadrada.

Conforme visto na seção 4.3, o tempo de uma multiplicação é de 31 ciclos de clock. Como são realizadas 16 multiplicações, o tempo gasto com as multiplicações são de 496 ciclos. Como o comparador é um Subtrator, como implementado no Divisor, o tempo de sua comparação tem a duração de um ciclo, totalizando 16 ciclos com esta operação. Assim, o tempo gasto com a Raiz Quadrada será de 512 ciclos, o que corresponde a 15.52 µs com um clock de 33 MHz.

4.5.2 - Raiz Quadrada: Método da Divisão

O cálculo da Raiz Quadrada por aproximações sucessivas através do método de NEWTON [SCA85], parte de um "chute" inicial A_0 e depois, sucessivas operações são

desencadeadas até atingir-se a precisão desejada. Este "chute" inicial é dado de acordo com a equação abaixo:

$$A_0 = \frac{N}{200} + 2$$

onde N é o número a se extrair a Raiz Quadrada.

As raizes aproximadas são obtidas a partir da expressão:

$$A_i = \frac{\frac{N}{A_{i-1}} + A_{i-1}}{2}$$

Para se encontrar a Raiz Quadrada de N, pode-se definir o Número de iterações máximas e/ou a variação mínima (precisão) entre as aproximações. Para exemplificar este método, fará-se-á a extração da Raiz Quadrada de 10000. O número máximo de iterações deve ser 7 e a precisão mínima de 10^{-3}. Assim:

Iteração	Aproximação	|Variação|
1ª	$A_0 = \frac{10000}{200} + 2 = 52.000$	------
2ª	$A_1 = \frac{\frac{10000}{52} + 52}{2} = 122.154$	$70.154 > 10^{-3}$
3ª	$A_2 = \frac{\frac{10000}{122.154} + 122.154}{2} = 102.009$	$20.145 > 10^{-3}$
4ª	$A_3 = \frac{\frac{10000}{102.009} + 102.009}{2} = 100.020$	$1.989 > 10^{-3}$
5ª	$A_4 = \frac{\frac{10000}{100.020} + 100.020}{2} = 100.000$	$0.020 > 10^{-3}$
6ª	$A_5 = \frac{\frac{10000}{100} + 100}{2} = 100.000$	$0.000 < 10^{-3}$

No exemplo acima, observa-se que a precisão desejada foi obtida na 6ª iteração.

Para implementação em hardware deste método, fica impraticável realizar uma operação de Raiz Quadrada monitorando a precisão, assim se faz necessário verificar o número mínimo de iterações que proporcione uma boa precisão. Resultados obtidos, indicam que um número de 10 iterações, permite uma precisão de 10^{-5}, o que é um valor satisfatório para o Transdutor Digital.

Este método é excelente para uso em software, mas apresenta uma grande desvantagem, a velocidade. Observa-se que, basicamente, este método se baseia em divisões, o que é possível de se realizar com a UAB. Assim, basta sequenciar as operações de carregamento dos operandos e selecionar as operações de divisão e adição, armazenando as aproximações em um registrador externo (presente na Unidade de Controle do Transdutor, por exemplo). Como na UAB, uma divisão por 2 não é realizada com um simples deslocamento à direita, implica na execução de uma divisão completa como outra qualquer. Assim, para extrair uma Raiz Quadrada com 10 iterações, são necessárias 20 divisões e 10 somas, o que consumiram 1290 ciclos de "clock" (1280 com as divisões e 10 com as adições). Para um "clock" de 33 MHz, o tempo necessário à realização da Raiz Quadrada é de 39.09 μs, cerca de 2.5 vezes maior que o método da multiplicação.

No projeto do Transdutor Digital, a operação de Raiz Quadrada será realizada com o método da multiplicação, pois além de mais rápido, permite sua implementação em um hardware mais simplificado. Como as operações necessárias à realização da Raiz (multiplicação e adição) são implementadas pela UAB, o hardware da Raiz será incluído na Unidade de Controle do Transdutor, requisitando as multiplicações e adições à UAB.

CAPÍTULO V

UNIDADE ARITMÉTICA BÁSICA

Com as bases do projeto montadas, pode-se agora desenvolver e implementar a UAB. Assim, neste capítulo será visto todo o procedimento adotado no projeto da UAB, desde sua epecificação lógica (comportamental) até seu "LAYOUT" final. Também será visto um algoritmo de testabilidade estudado e o procedimento adotado no teste da UAB.

5.1 - Interface da UAB

Num projeto "VLSI", deve-se caracterizar (definir) bem o projeto a ser desenvolvido (ver seção 3.3), para facilita-lo e evitar alterações no final deste, pois o custo de correções cresce exponencialmente com o avanço das fases do projeto.

Para se definir a interface para a UAB, relacionam-se primeiro as características do projeto. Depois cada característica é estudada para expressar fisicamente sua funcionalidade. Suas cinco características são:

- Manipular operandos de 32 bits,
- Utilizar um barramento de entrada/saída de 16 bits e
- realizar operações de Soma, Subtração, Multiplicação e Divisão,
- A seleção deve ser feita em pinos específicos (instruções),
- Possuir um sinal para início e fim de operação.

Agora cada característica será analisada, de tal forma que se possa definir o diagrama de tempo completo da UAB.

característica 1 - A utilização de operandos de 32 bits, implica no uso de registradores do mesmo tamanho. Como a base do projeto é um Somador, conforme visto no capítulo 4, este deve ser de 32 bits (seção 4.2.1).

característica 2 - Como em um projeto de circuito integrado, a área se silício necessária à sua realização é de suma importância, o barramento deverá ser de 16 bits. Além disso, deverá ser de entrada e saída. Isto porque um dos fatores que mais contribuem para o crescimento da área de um chip é o número de pinos, pois os "PADS" ocupam uma área muito grande (ver SCLIB). Como os operandos são de 32 bits, deverá existir um controle na entrada e saída para esta compatibilização. No caso da entrada, deverá existir um pino (LOAD) que faz o carregamento de blocos de 16 bits até completar os dois operandos. Na saída deve existir outro pino (SEL_OUT) que seleciona os 16 bits menos significativos e os 16 mais significativos.

característica 3 - A UAB deverá implementar as operações de Adição, Subtração. Multiplicação e divisão em 32 bits com base nos algoritmos apresentados no capítulo 4.

característica 4 - As operações aritméticas serão selecionadas por pinos (o0 e o1) específicos, que definem as quatro instruções da UAB (seção 5..2.2).

característica 5 - Um sinal de início de operação (OPERA) deve ser dado sempre que os operandos já estejam armazenados nos seus respectivos registradores e uma instrução esteja nos pinos de seleção de operação. Ao final da operação selecionada, um pino (FIM) deverá informar o término desta.

Assim, a interface da UAB terá a organização mostrada na figura 5.1.

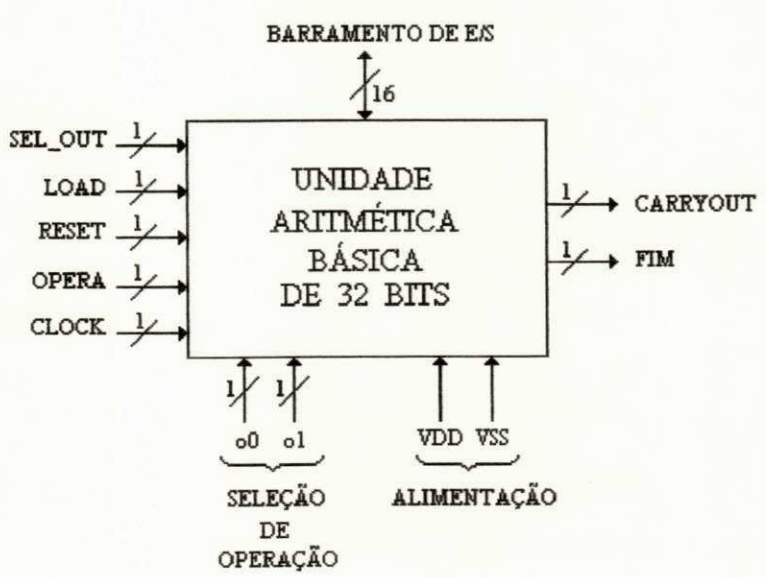


Figura 5.1 - Interface da UAB

5.2 - Diagramas de Tempo

Com a definição da interface da UAB, pode-se caracterizar o circuito do ponto de vista funcional. Para isso, faz-se o traçado de todos os sinais em diagrama de tempo. Assim, fica definido o que ocorre a cada instante (cada ciclo de clock) com os sinais do circuito.

Para traçar o diagrama de tempo da UAB, divide-se a sequência de uma operação em três fases: carregamento dos operandos; execução da operação e seleção de saída. Assim, em cada fase será discutido, em detalhes, todo o hardware da UAB.

5.2.1 - Carregamento dos Operandos

Como os operandos são de 32 bits e o barramento de 16 bits, precisa-se de um bloco que faça a distribuição correta dos dados. Como na UAB existem dois operandos, serão precisos quatro carregamentos de blocos de 16 bits, para o preenchimento dos registradores.

A sequência de carregamento deve ser bem definida, para que os dados sejam colocados no barramento na ordem correta. Para isso, divide-se cada operando de 32 bits em duas palavras de 16 bits: palavra menos significativa (LSW) e a palavra mais significativa (MSW). A figura 5.2 mostra esta divisão.

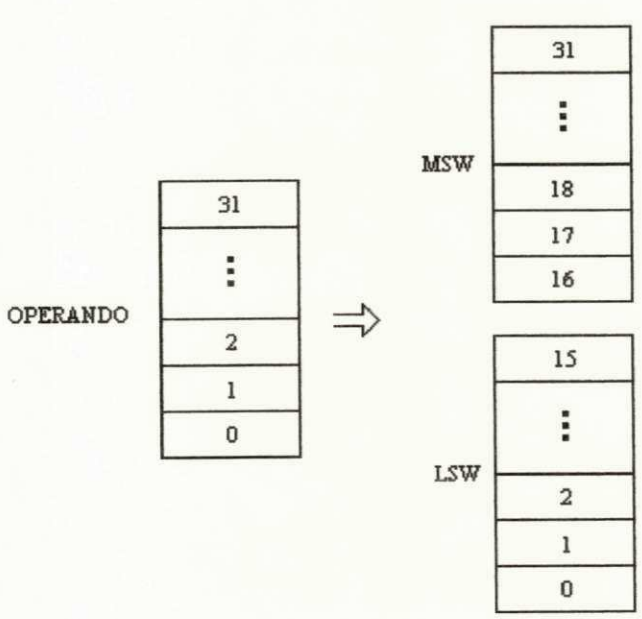


Figura 5.2 - Divisão dos Operandos em Duas Palavras de 16 Bits

Com isso, pode-se definir a sequência de carregamento:

- Carregamento 1 - LSW do primeiro operando (LSW 1),
- Carregamento 2 - MSW do primeiro operando (MSW 1),
- Carregamento 3 - LSW do segundo operando (LSW 2) e
- Carregamento 4 - MSW do segundo operando (MSW 2),

Definida a sequência de carregamento dos operandos, pode-se traçar o diagrama de tempo desta fase, ver figura 5.3.

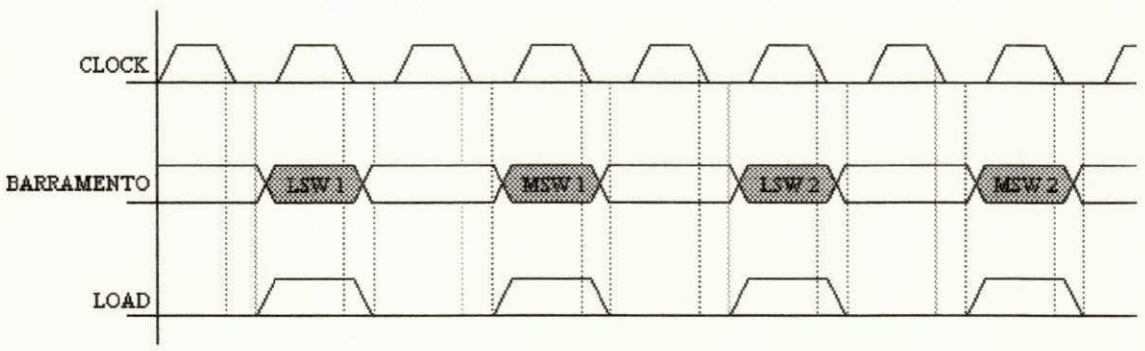


Figura 5.3 - Diagrama de Tempo do Carregamento

A figura 5.4 mostra o diagrama de blocos desta fase. O bloco "Seleção de Operandos" é composto por quatro mux 16 x 2:1 e o clock utilizado nos dois registradores é o mesmo. Assim, uma das entradas vem do barramento e a outra da própria saída de 16 bits correspondente. Desta forma, quando vier um pulso de clock ele, ou carrega o valor do barramento, ou mantém o mesmo valor (a figura 5.5 mostra esta operação). Esta operação é comandada pelo bloco "Controle", que consta de um contador de 2 bits, onde cada estado corresponde ao carregamento de uma palavra em seu respectivo registrador. O clock deste contador é controlado pelo sinal LOAD, ver figura 5.6. Observe que o contador só troca de estado quando LOAD cai a "0", pois a saída do contador só se altera quando cks (clock escravo) está em "1". Nesta figura ainda é vista a lógica de geração de c1, c2, c3 e c4.

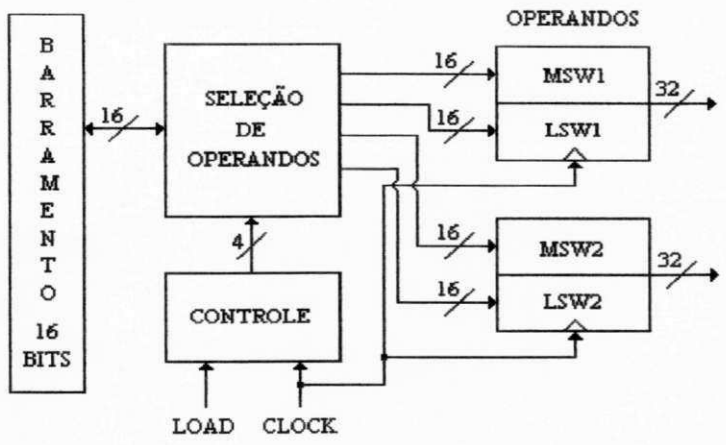


Figura 5.4 - Diagrama de Blocos do Circuito de Carregamento

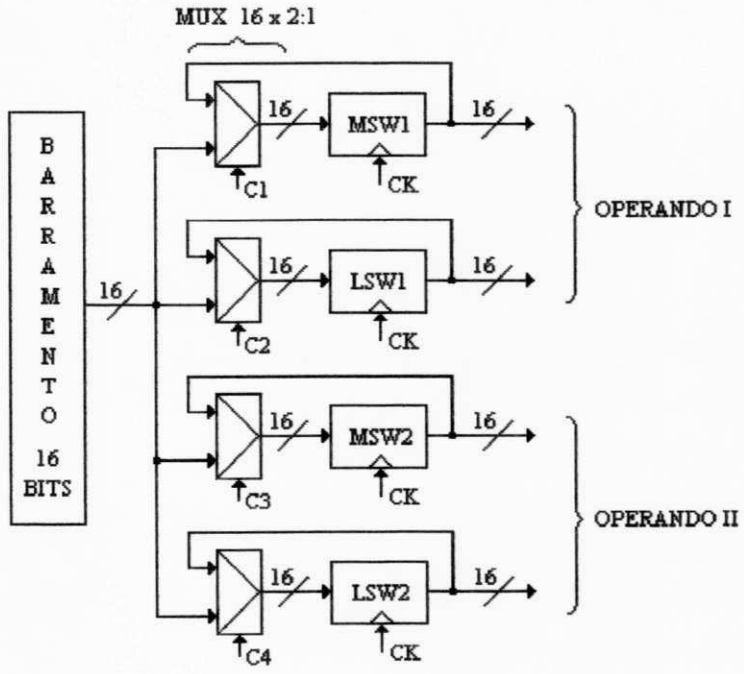


Figura 5.5 - Mux 16 x 2:1 do Bloco "Seleção de Operandos"

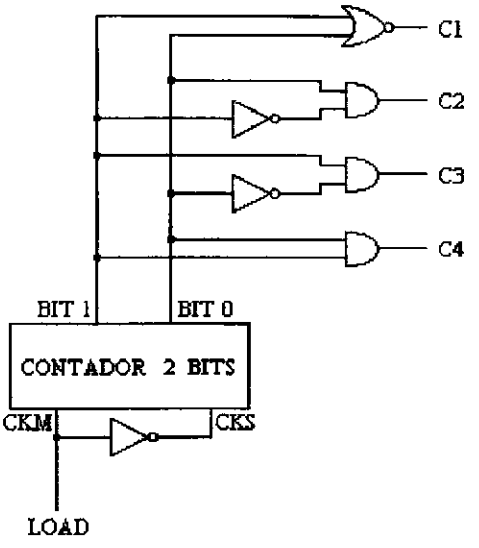


Figura 5.6 - Contador do "Controle"

5.2.2 - Execução da Operação

Com o carregamento dos operandos em seus respectivos registradores, pode-se dar início a uma operação aritmética. Para isso, define-se a operação através dos pinos o0 e o1, ver tabela 5.1, e dá início a execução da operação através de um pulso no pino OPERA. Com isso, o contador dos ciclos de clock necessários para a realização de uma operação (no caso da multiplicação e divisão) é resetado, levando o pino FIM a "1" indicando que uma operação está em andamento. Quando o contador atingir 31 (multiplicação) ou 32 (divisão), este pino volta para "0", indicando fim de operação. No caso da soma e subtração, que são paralelas, este pino não sofre alteração, pis ao final do pulso em OPERA, o resultado já estará disponível no próximo ciclo de clock. A figura 5.7 mostra o diagrama de tempo durante a execução de uma operação aritmética.

Pinos		Operação
o1	o0	
0	0	Multiplicação
0	1	Adição
1	0	Divisão
1	1	Subtração

Tabela 5.1 - Definição das Instruções da UAB

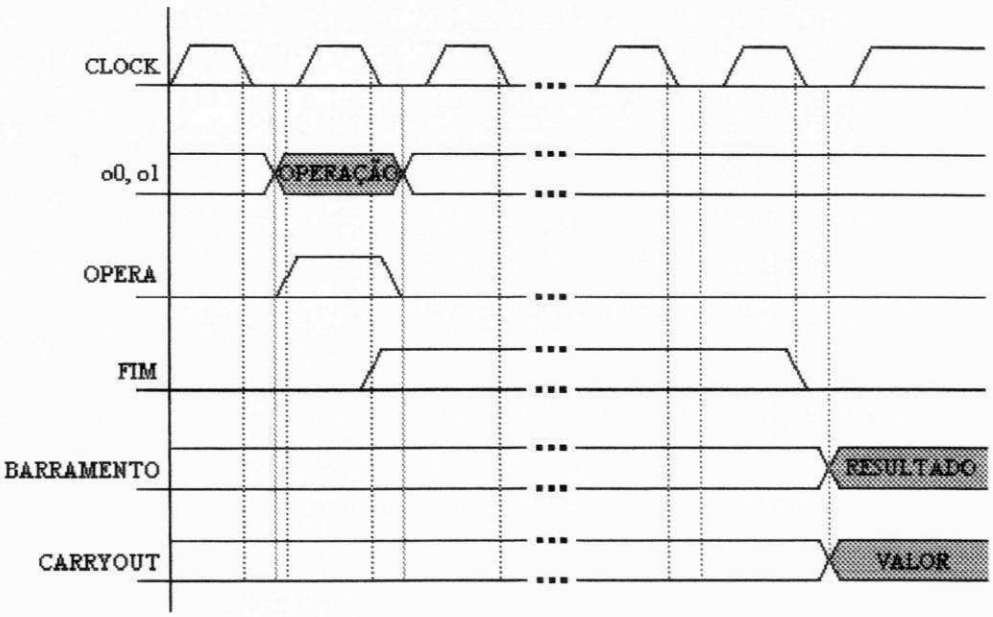


Figura 5.7 - Diagrama de Tempo da Execução de uma Operação Aritmética

A figura 5.8 mostra o diagrama de blocos do circuito de execução de operações aritméticas. O bloco "coração" será visto em detalhe na seção 5.3. Observe que existem duas saídas de resultados, uma para a multiplicação/divisão e a outra para soma/subtração. Isso porque o resultado de soma/subtração não são armazenados em registradores, o resultado é obtido diretamente do Somador, ao contrário da multiplicação/divisão que compõem seus resultados em registradores específicos.

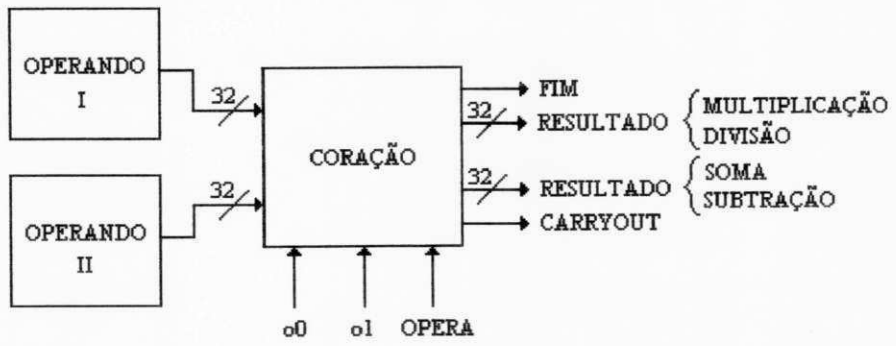


Figura 5.8 - Diagrama de Blocos do Circuito de Execução de Operações Aritméticas

5.2.3 - Seleção de Saída

Esta fase é semelhante a de "Seleção de Operandos", ou seja, divide-se blocos de 32 bits em outros de 16 bits. Como existem dois blocos de 32 bits para os resultados, deve-se

selecionar primeiro um dos blocos e depois selecionar as palavras de 16 bits a ser colocadas no barramento.

Pela figura 5.8, observa-se os dois de 32 bits utilizados para os resultados. Como as operações são selecionadas pelos pinos o0 e o1, automaticamente um dos blocos serão ativados de acordo com a operação. Observe pela tabela 5.1 que quando o0 está em "0", a operação é multiplicação/divisão e em "1" é soma/subtração. Assim este pino será usado neste controle. Com isso, o único controle externo é feito pelo pino SEL_OUT onde é escolhida a palavra de 16 bits a ser colocada no barramento. No projeto, adotou-se que quando SEL_OUT estiver em "0" será apresentado o LSW, e em "1" o MSW. A figura 5.9 apresenta o diagrama de tempo da seleção de saída, e a figura 5.10 o circuito.

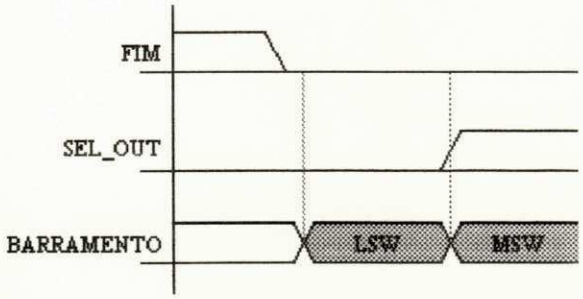


Figura 5.9 - Diagrama de Tempo da Seleção de Saída

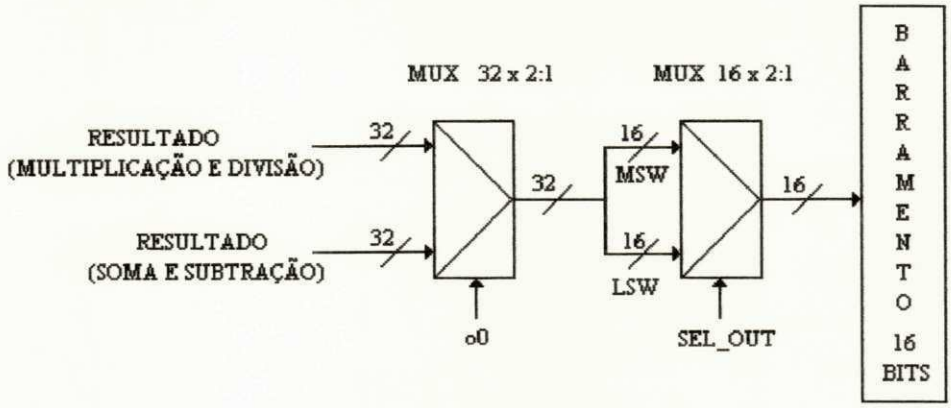


Figura 5.10 - Circuito de Seleção de Saída

5.3 - Implementação da UAB

Com a caracterização de todo o projeto da UAB, pode-se implementar o hardware desta unidade. Com os procedimentos adotados nas seções anteriores, pode-se definir o diagrama de blocos da UAB, é o que mostra a figura 5.11.

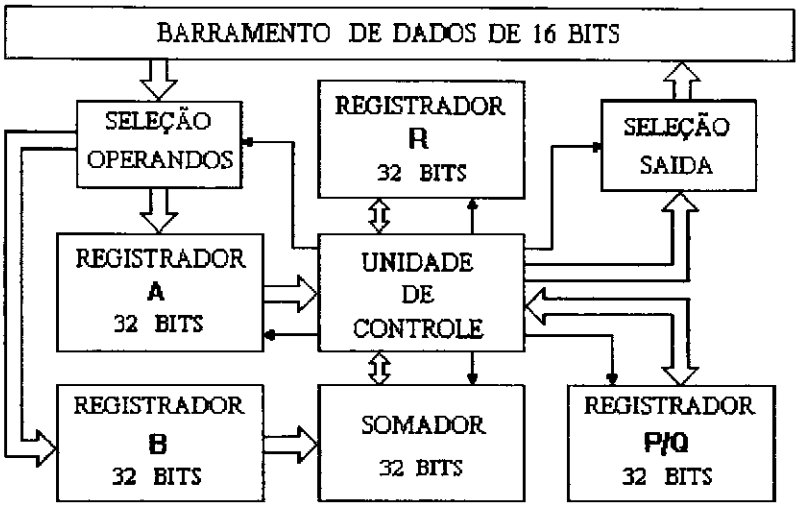


Figura 5.11 - Diagrama de Blocos da UAB

Os blocos "Seleção de Operandos" e "Seleção de Saída" foram vistos nas seções 5.2.1 e 5.2.3, respectivamente. O Somador foi mostrado na seção 4.2.1. Os registradores A, B, R e P/R e a unidade de controle serão apresentados nesta seção. Para isso, serão feitas comparações entre o Multiplicador e o Divisor, apresentados no capítulo 4, de forma a compatibilizar as duas operações em um único circuito.

5.3.1 - Registrador A (1º operando)

Analisando o Multiplicador e o Divisor, vistos no capítulo 4, pode-se ver, pelas figuras 4.11 e 4.16, que os operandos multiplicador e dividendo, constam de registradores de deslocamento. No caso do primeiro, o deslocamento é à direita, e o segundo à esquerda. Na seção 5.2.1 viu-se que os registradores dos operandos recebem carregamentos paralelos, isto também pode ser visto nas figuras citadas acima. Assim, para satisfazer aos dois circuitos, Multiplicador e Divisor, precisa-se de um registrador de carregamento paralelo e deslocamentos à direita e esquerda. Este registrador deve possuir uma entrada de "reset", para permitir a "limpeza" total deste para a realização de uma operação aritmética. A figura 5.12 mostra o circuito do registrador A de 32 bits.

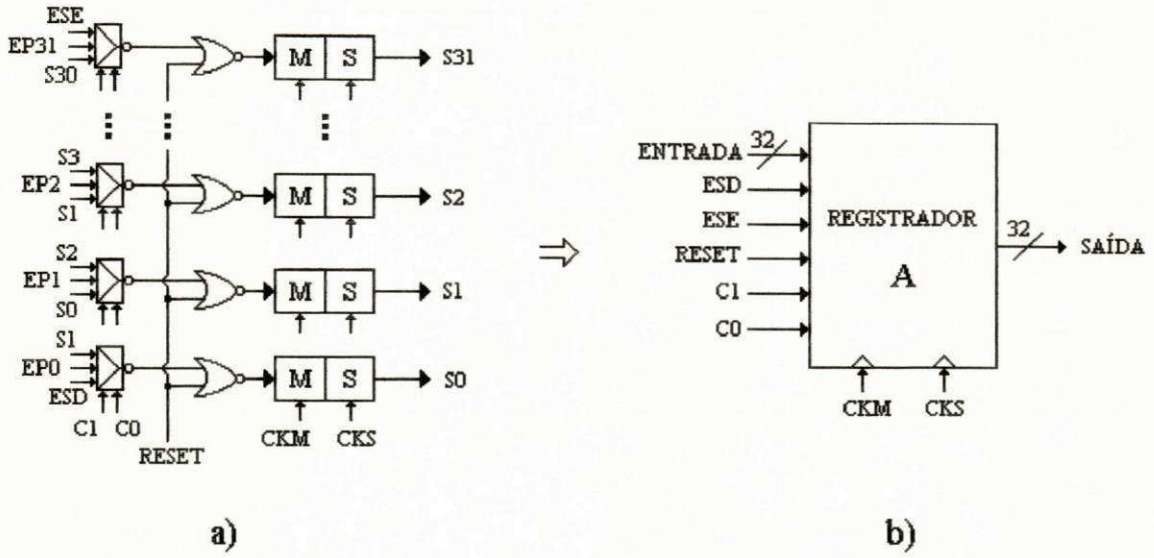


Figura 5.12 - Registrador A: a) Circuito, b) Interface

Quando "reset" vai a "1", com um ciclo de clock o registrador conterá "0" em todos os seus bits. Quando "reset" está em "0", é habilitada a passagem de uma das entradas do registrador, de acordo com a seleção feita em C1 e C0, ver tabela 5.2. Quando o modo de deslocamento à direita é ativado, os bits que entrarão no registrador devem ser colocados no pino ESE (Entrada Serial à Esquerda), e se o deslocamento for à esquerda, o pino utilizado é o ESD (Entrada Serial à Direita).

Controle		Modo de Operação
C1	C0	
0	0	deslocamento à esquerda
0	1	carregamento paralelo
1	0	deslocamento à direita
1	1	deslocamento à direita

Tabela 5.2 - Modo de Operação do Registrador A

No caso específico da UAB, neste registrador os pinos ESD e ESE devem estar em "0". Os controles C1 e C0 devem ser calculados de acordo com a operação. No caso do carregamento dos operandos, este controle deve selecionar o modo de carregamento paralelo "01", ou seja, depende do sinal LOAD. Quando a operação for multiplicação, o controle deve ser "10" ou "11", deslocamento á direita, e "00" no caso da divisão. Assim, pela tabela 5.3.

LOAD	o1	o0	C1	C0
0	0	0	1	x
0	0	1	x	x
0	1	0	0	0
0	1	1	x	x
1	0	0	0	1
1	0	1	0	1
1	1	0	0	1
1	1	1	0	1

Tabela 5.3 - Definição de C1 e C0

Assim,

$C_1 = \overline{LOAD + o1}$ e

$C_0 = LOAD$

Este controle é uma lógica externa ao registrador, assim ele será incluído na unidade de controle da UAB vista na seção 5.3.6.

5.3.2 - Registrador B (2º Operando)

Este é o registrador do segundo operando, ou seja, multiplicando e divisor. Pelas figuras 4.11 e 4.16, observa-se que este é um registrador apenas de carregamento paralelo. No caso do divisor, ele é inversor, e no caso do multiplicando ele faz uma lógica do tipo "AND". Este inversor e o "AND", podem sair deste registrador e entrar na Unidade de Controle da UAB. A figura 5.13 mostra o circuito do registrador B com "reset".

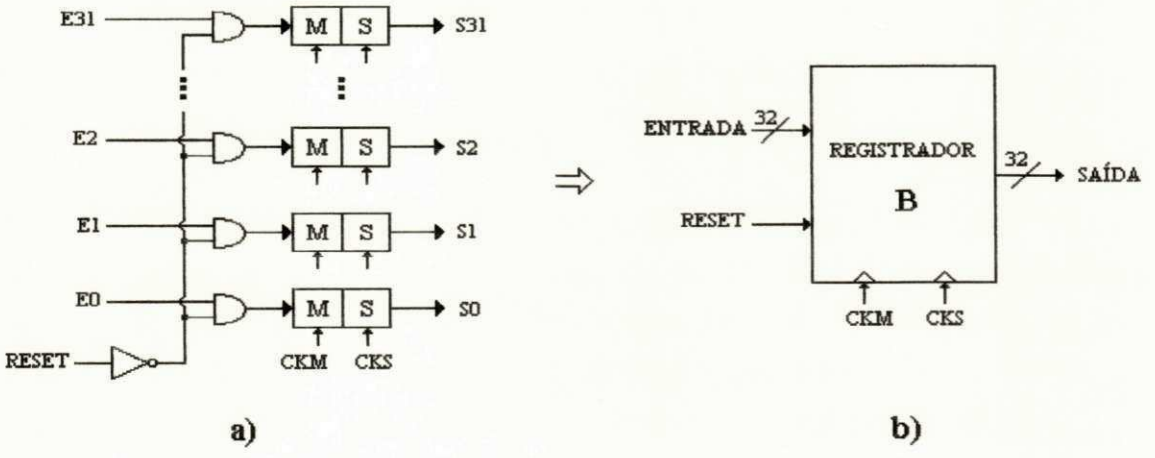


Figura 5.13 - Registrador B: a) Circuito, b) Interface

Na lógica externa que deve complementar este registrador, inversor e "AND", deve-se levar em consideração, nas operações de soma e subtração, que este operando não deve sofrer nenhuma alteração, assim, essa lógica deve possuir mais um estado, que é a transparência, ou seja, a saída igual à entrada. O circuito da figura 5.14 mostra tal lógica, que deve integrar a unidade de controle da UAB.

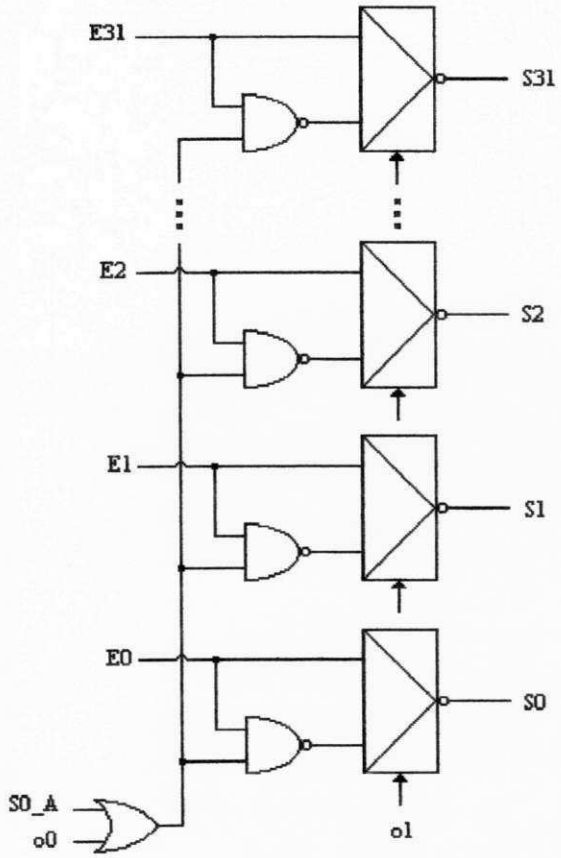


Figura 5.14 - Lógica Adicional do Registrador B

Analisando o circuito observa-se que, quando o1 está em "1", indicando divisão ou subtração, S recebe E invertido, pois o mux é inversor. Quando o1 está em "0", indicando multiplicação ou soma, S recebe o valor de "E AND S0_A" (saída 0 do registrador A), pois como o mux é inversor e com o "NAND", tem-se uma operação de "AND". Esta operação é desejada na multiplicação, ou seja, com o0 em "0". Quando o0 está em "1", indicando soma, tem-se o "NAND" operando como inversor. Com o mux inversor, tem-se S=E. Com isso, a lógica está definida.

5.3.3 - Registrador R (Resto)

Este é um registrador de deslocamento à esquerda e carregamento paralelo. No projeto da UAB, foi utilizado o mesmo registrador A, só que programado para os dois casos citados. A figura 5.15 mostra a forma como ele foi utilizado.

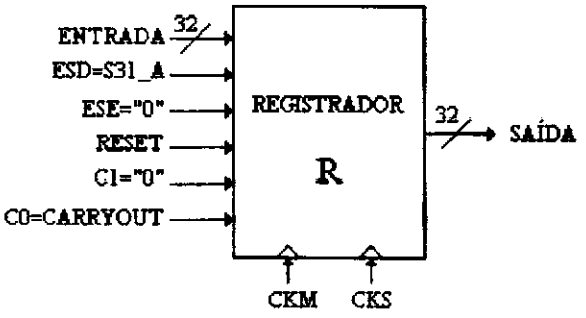


Figura 5.15 - Registrador A Configurado para R

5.3.4 - Registrador P/Q (Produto/Quociente)

Este registrador quando opera como produto, realiza deslocamentos à direita, e como quociente, desloca à esquerda, além de fazer carregamentos paralelos. Assim, observa-se que este registrador é idêntico ao A.

5.3.5 - Somador

O Somador da UAB foi desenvolvido no capítulo 4. Precisa-se determinar a lógica de seleção dos dados a entrarem nele. Esta lógica será incluída na Unidade de Controle.

No caso da soma e subtração, as entradas do somador são o primeiro e o segundo operandos. Na multiplicação, pela figura 4.11, as entradas são o produto e o segundo operando. Na divisão, são o segundo operando e o resto.

Observe que o segundo operando sempre está presente em uma das entradas do Somador. Mas isto é feito através da lógica definida na seção 5.3.2. No caso da outra entrada, deve-se colocar um mux para fazer a seleção entre o primeiro operando, o resto e o produto. A figura 5.16 mostra o diagrama de blocos das ligações do Somador na UAB.

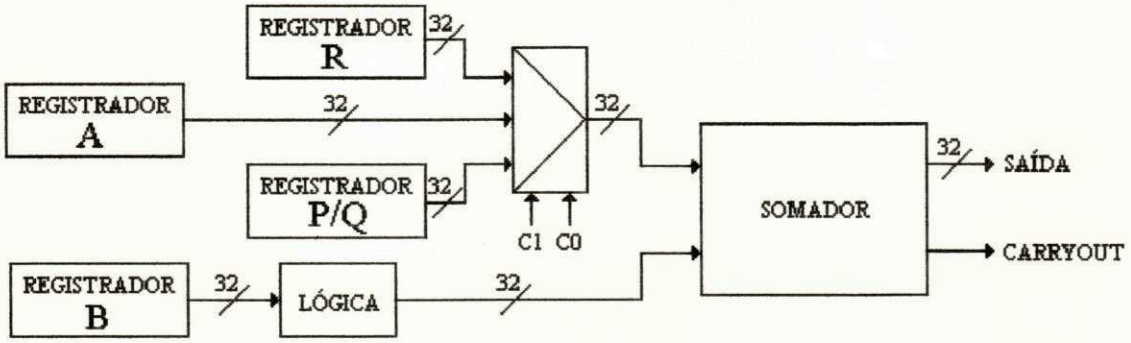


Figura 5.16 - Conexão do Somador

O mux do Somador será incluído na unidade de controle da UAB, bem como a lógica de seu controle, mostrada na tabela 5.4.

o1	o0	C1	C0	Saída
0	0	0	0	Reg P/Q
0	1	0	1	Reg A
1	0	1	0	Reg R
1	1	0	1	----------

Tabela 5.4 - Tabela da Verdade do Mux do Somador

Assim,

$$C_1 = o1.\overline{o0}$$

e $$C_0 = o0$$

5.3.6 - Unidade de Controle

Esta unidade é responsável pelo perfeito funcionamento da UAB. É ela quem fornece os sinais correspondentes as operações aritméticas requisitadas. por isso, toda lógica combinacional da UAB está nela contida.

Uma parte da Unidade de Controle já foi apresentada nas seções anteriores. A parte restante se refere ao controle dos clocks dos registradores. Para se implementar esta parte, simplesmente uniu-se os controles dos circuitos do Multiplicador (figura 4.11) e o Divisor (figura 4.16). O circuito final obtido está mostrado na figura 5.17.

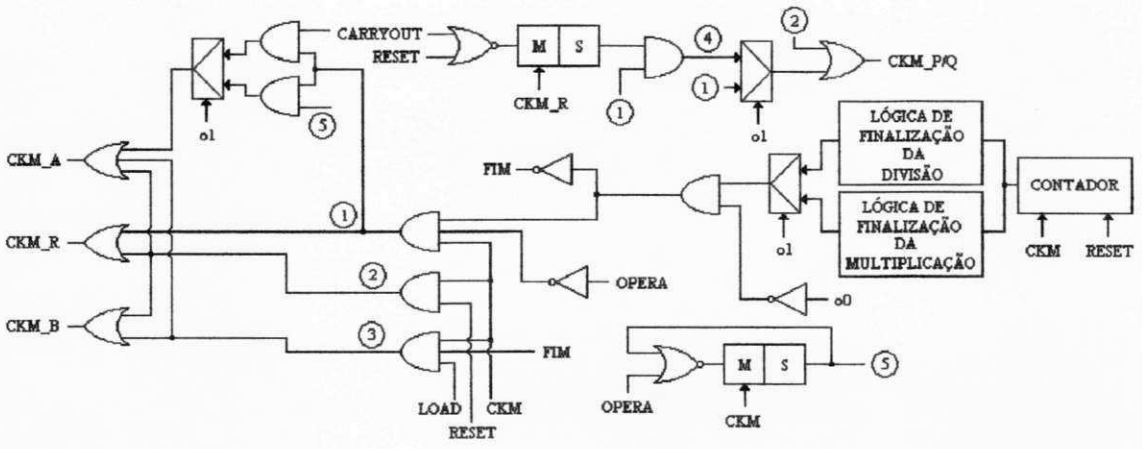


Figura 5.17 - Controle dos Clocks da UAB

A descrição comportamental da UAB está mostrada no ANEXO IV.

5.4 - Testabilidade

Nos projetos "VLSI", as validações são feitas através dos vetores de teste, onde submetidos às análises de um simulador (ASIMUT no ALLIANCE), informa se os valores impostos nas saídas estão ou não de acordo com a descrição comportamental do circuito.

Para se obter os vetores de teste, basta montar a tabela da verdade do circuito. Cada linha da tabela forma um vetor. Assim, os valores das entradas são fornecidas ao simulador e o resultado obtido é comparado com o do vetor de teste. Se o resultado coincidir, o circuito está validado para este vetor. Com a validação de todos os vetores de teste, tem-se também a do circuito.

Conforme visto no capítulo 3, um Somador de 32 bits, possui $1{,}845.10^{19}$ vetores de teste. Este é um número muito grande e precisa ser reduzido, pois quando se parte para os testes do Circuito Integrado na fábrica, o tempo de teste deve ser o menor possível. Para isso,

existem os algoritmos de testabilidade de Circuitos Integrados, onde se obtém o menor número possível de vetores que o validem o máximo possível.

O algoritmo D [RBS67] é um dos mais simples e eficientes na escolha dos vetores de teste. Para explicar este algoritmo, considera-se o circuito da figura 5.18 e sua tabela da verdade (tabela 5.5).

$$F = A.B + \overline{A + C} + BD$$

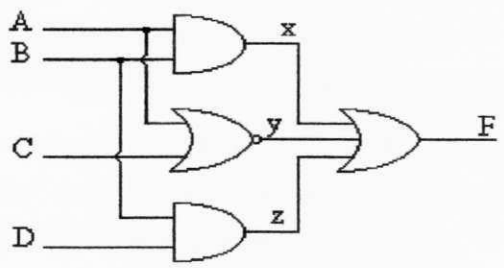


Figura 5.18 - Circuito para Teste com o Algoritmo D

VETOR	ENTRADAS				SINAIS			SAÍDA
	A	B	C	D	x	y	z	F
0	0	0	0	0	0	1	0	1
1	0	0	0	1	0	1	0	1
2	0	0	1	0	0	0	0	0
3	0	0	1	1	0	0	0	0
4	0	1	0	0	0	1	0	1
5	0	1	0	1	0	1	1	1
6	0	1	1	0	0	0	0	0
7	0	1	1	1	0	0	1	1
8	1	0	0	0	0	0	0	0
9	1	0	0	1	0	0	0	0
10	1	0	1	0	0	0	0	0
11	1	0	1	1	0	0	0	0
12	1	1	0	0	1	0	0	1
13	1	1	0	1	1	0	1	1
14	1	1	1	0	1	0	0	1
15	1	1	1	1	1	0	1	1

Tabela 5.5 - Tabela da Verdade do Circuito da Figura 5.18

Os vetores de teste escolhidos, servirão para testar o Circuito Integrado já difundido. Como não há tempo para testa-lo integralmente, estes vetores devem testar se existem sinais internos ao chip, que estejam "colados" em "0" ou em "1". Assim, o objetivo do algoritmo D é identificar os vetores de teste que satisfazem estas condições.

CAPÍTULO VI

CONCLUSÕES E SUGESTÕES

A Unidade Aritmética Básica (UAB) desenvolvida, realiza operações de Soma, Subtração, Multiplicação e Divisão binárias em 32 bits, exceto a Multiplicação. Os resultados das simulações com o ASIMUT (simulador lógico do sistema ALLIANCE) foram satisfatórios, apresentando boa precisão, especialmente no caso da Divisão, onde blocos de 32 bits podem ser obtidos através do pino OPERA. Quando uma operação de Divisão é solicitada, tem-se ao final, um número de 32 bits. Se o registrador R contém o valor residual de algum resto de Divisão, através de mais uma solicitação de Divisão com o pino OPERA, terá-se-á mais 32 bits decimais de precisão. Este procedimento pode ser repetido indefinidamente se necessário. Assim, pode-se dizer que o Divisor da UAB possui precisão infinita.

No projeto original do Transdutor Digital, tanto os dados de entrada, oriundos dos A/D's, quanto os resultados, são fornecidos em 10 bits. Com a UAB desenvolvida, os resultados poderão atingir uma precisão de 32 bits, possibilitando ainda, o uso de A/D's de melhor precisão.

A partir dos dados da documentação da SCLIB, o desempenho estimado para UAB permitirá operar com uma frequência de 33 MHz. Nessas condições, o transdutor digital será capaz de realizar pelo menos 660 amostras por período.

A área total ocupada pelo circuito, incluindo a coroa de "PADS", foi de 18,27 mm^2 (3,98 mm X 4,59 mm, incluindo 1200 células com 9828 transistores), figura 6.1. O uso da metodologia de projetos "STANDARD CELLS", levou a um gasto de cerca de 3/4 desta área com roteamento (canais), inviabilizando a integração no mesmo chip, da Unidade de Controle, com vistas ao PMU.

Com a inclusão da tecnologia DPLIB (que permite o roteamento sobre as células) no sistema ALLIANCE em princípios de 1994, o ganho de área dos projetos "VLSI" será bem acentuado. Estimativas (fornecidas pelo laboratório MASI) revelam que com o uso da DPLIB, a área da UAB se reduzirá a aproximadamente 5 mm^2, o que corresponde a uma redução de 72%.

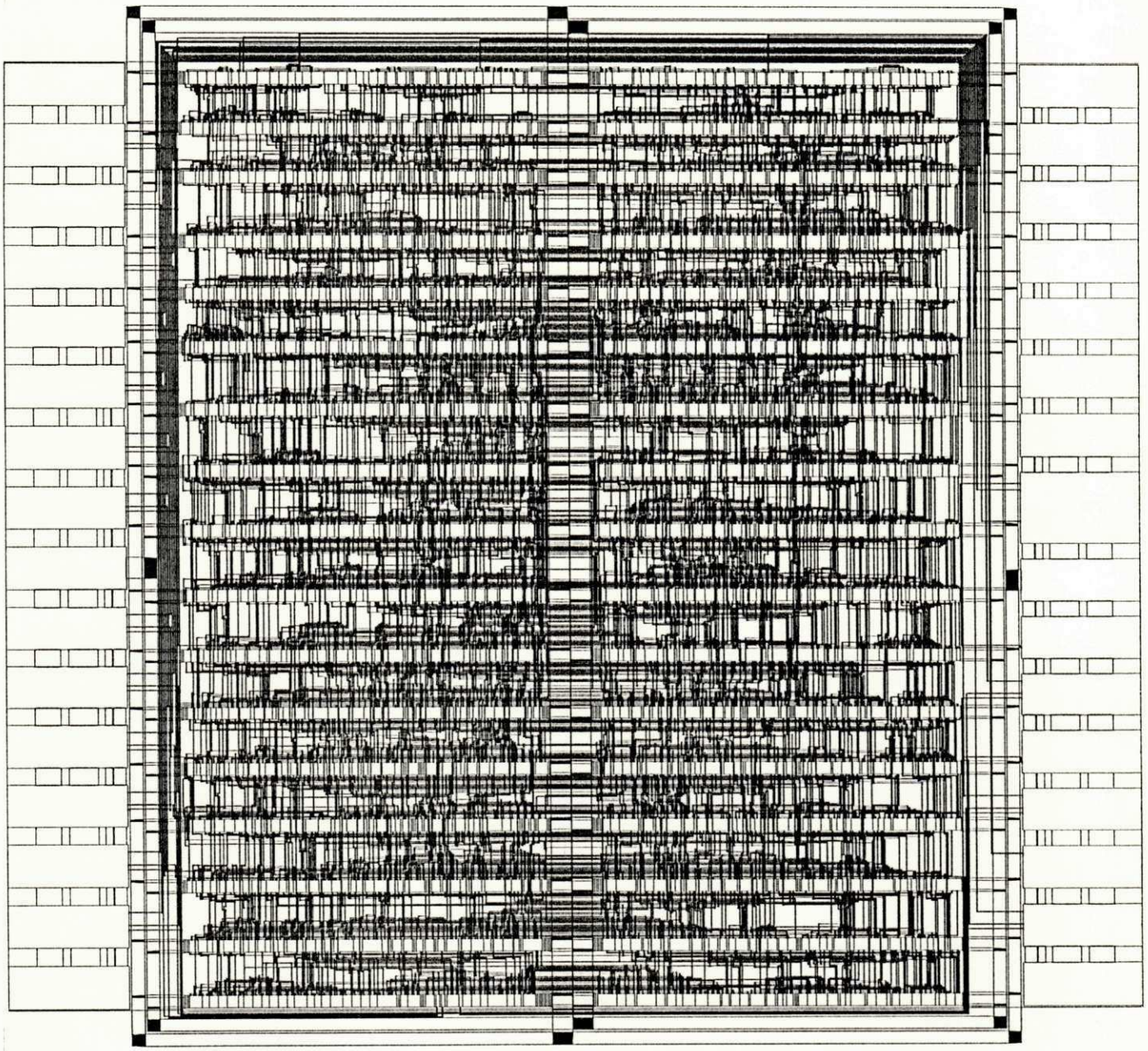


Figura 6.1 - Layout da UAB

6.1 - Sugestões

Para aumentar a velocidade no carregamento dos operandos, especialmente no caso da operação de quadrado com o Multiplicador, sugere-se a inclusão de um pino na UAB, indicando se os operandos são de 16 ou de 32 bits. No caso de se selecionar os 32 bits, o carregamento se dá da forma apresentada no capítulo 5. No caso de operandos de 16 bits, logo após um RESET (todos os registradores em "0"), um sinal de LOAD em "1" carrega o operando de 16 bits presente no barramento nos LSW1 e LSW2. Isto possibilita a seleção de quadrado através dos pinos o1 e o0 ("00" multiplicação), seguida do sinal de OPERA. Neste caso, uma operação foi realizada a partir de um único LOAD.

Prosseguindo com o segundo LOAD, o operando será armazenado em LSW2, permitindo uma multiplicação de 16 bits inteira positiva, com dois LOAD'S.

Uma outra vantagem desta alteração é que, com dois LOAD'S, tem-se dois operandos de 16 bits nos registradores A e B, permitindo, não só operações de multiplicação, mas também de soma, subtração e divisão, sem necessidade de quatro ciclos de LOAD.

A respeito da multiplicação, como o Somador da UAB é de 32 bits e durante a realização desta operação, o registrador R não é utilizado, sugere-se uma ligeira alteração no hardware da UAB, onde este registrador é acoplado ao P/Q tornando-se um registrador de 64 bits. Isso possibilita multiplicações em 32 bits com operandos inteiros positivos.

No caso da multiplicação de 16 bits, a sequência de operação do registrador P/Q, é sempre um carregamento paralelo seguido de um deslocamento à direita, Exigindo dois ciclos de "clock", totalizando 31 ciclos para toda a operação (a última operação deste registrador é um carregamento paralelo). Sugere-se que estes dois passos sejam feitos em um único ciclo de "clock", ou seja, os valores já são carregados com seus bits deslocados à direita. Com isso, consegue-se uma multiplicação com 16 ciclos de relógio, o que proporcionará o dobro de velocidade. Esta mesma sugestão é válida para o registrador R do Divisor, com isso consegue-se divisões fixas com 32 ciclos de "clock" e não 64 como no pior caso (ver capítulo 4).

Como a UAB é uma unidade independente e flexível, ela pode ser utilizada com outros sistemas. Assim, para se obter o máximo das operações aritméticas, sugere-se estudar os algoritmos da multiplicação e divisão, de tal forma que se possa trabalhar com aritmética de sinal (complemento a dois). Simulações realizadas mostram que no multiplicador, se os operandos negativos são complementados a dois em 32 bits, o resultado será fornecido também em complemento a dois. Como esta aritmética é suficiente para o Transdutor Digital, não foram feitas simulações com o Divisor com números negativos, ficando portanto, como sugestão.

Com relação a testabilidade da UAB, sugere-se a inclusão da metodologia "SCAN PATH" para os testes da lógica sequencial e combinacional entre os registradores. Isto inclui mais dois pinos ao chip, são o SC_{in} (SCAN IN) e o SC_{out} (SCAN OUT). Estes pinos são a entrada e saída dos bits de teste da lógica sequencial.

Para o ganho de área de silício, citado nas conclusões, sugere-se a passagem do "LAYOUT" da UAB para a tecnologia DPLIB, pois como foi visto, a redução será bem significativa.

Finalmente, cabe ressaltar que tais sugestões já poderão ser incorporadas ao projeto do Transdutor Digital global, pois com a metodologia adotada ("TOP-DOWN"), qualquer parte do sistema pode ser modificada independentemente, desde que respeitadas as especificações de interface.

ANEXO I

SOMADOR

```
--=============================================================
-- Descricao Comportamental do bloco SOMADOR.VBE
--=============================================================

entity somador is

        port (
                entrada1   : in  bit_vector(0 to 31) ;
                entrada2   : in  bit_vector(0 to 31) ;
                carryin    : in  bit ;
                saida      : out bit_vector(0 to 31) ;
                carryout   : out bit ;
                vdd        : in  bit ;
                vss        : in  bit
        ) ;

end somador ;

architecture comportamental of somador is

-------------------------------------
-- Sinal do bloco Somador
-------------------------------------

        signal c     : bit_vector(0 to 31) ;  -- carrys da soma de cada bit
        signal p     : bit_vector(0 to 31) ;
        signal g     : bit_vector(0 to 31) ;
        signal pp    : bit_vector(0 to 9)  ;
        signal gg    : bit_vector(0 to 9)  ;

begin

        assert (((vdd xor '1') or (vss xor '0')) = '0')
                report "Alimentacao perdida em SOMADOR" severity warning;
-----------------------------------------------------------
-- geracao dos carry propagate/generate
-----------------------------------------------------------

        p <= entrada1 xor entrada2 ;
        g <= entrada1 and entrada2 ;
```

```
--------------------------------
-- descricao do BCLA
--------------------------------

c(0)  <= g(0)  or (p(0)  and carryin) ;
c(1)  <= g(1)  or (p(1)  and g(0))  or (p(0)  and p(1)  and carryin) ;
c(2)  <= g(2)  or (p(2)  and g(1))  or (p(1)  and p(2)  and g(0))  or (p(0)  and p(1)  and p(2)  and carryin) ;
c(4)  <= g(4)  or (p(4)  and c(3)) ;
c(5)  <= g(5)  or (p(5)  and g(4))  or (p(4)  and p(5)  and c(3)) ;
c(6)  <= g(6)  or (p(6)  and g(5))  or (p(5)  and p(6)  and g(4))  or (p(4)  and p(5)  and p(6)  and c(3)) ;
c(8)  <= g(8)  or (p(8)  and c(7)) ;
c(9)  <= g(9)  or (p(9)  and g(8))  or (p(8)  and p(9)  and c(7)) ;
c(10) <= g(10) or (p(10) and g(9))  or (p(9)  and p(10) and g(8))  or (p(8)  and p(9)  and p(10) and c(7)) ;
c(12) <= g(12) or (p(12) and c(11)) ;
c(13) <= g(13) or (p(13) and g(12)) or (p(12) and p(13) and c(11)) ;
c(14) <= g(14) or (p(14) and g(13)) or (p(13) and p(14) and g(12)) or (p(12) and p(13) and p(14) and c(11)) ;
c(16) <= g(16) or (p(16) and c(15)) ;
c(17) <= g(17) or (p(17) and g(16)) or (p(16) and p(17) and c(15)) ;
c(18) <= g(18) or (p(18) and g(17)) or (p(17) and p(18) and g(16)) or (p(16) and p(17) and p(18) and c(15)) ;
c(20) <= g(20) or (p(20) and c(19)) ;
c(21) <= g(21) or (p(21) and g(20)) or (p(20) and p(21) and c(19)) ;
c(22) <= g(22) or (p(22) and g(21)) or (p(21) and p(22) and g(20)) or (p(20) and p(21) and p(22) and c(19)) ;
c(24) <= g(24) or (p(24) and c(23)) ;
c(25) <= g(25) or (p(25) and g(24)) or (p(24) and p(25) and c(23)) ;
c(26) <= g(26) or (p(26) and g(25)) or (p(25) and p(26) and g(24)) or (p(24) and p(25) and p(26) and c(23)) ;
c(28) <= g(28) or (p(28) and c(27)) ;
c(29) <= g(29) or (p(29) and g(28)) or (p(28) and p(29) and c(27)) ;
c(30) <= g(30) or (p(30) and g(29)) or (p(29) and p(30) and g(28)) or (p(28) and p(29) and p(30) and c(27)) ;

pp(0) <= p(0)  and p(1)  and p(2)  and p(3) ;
gg(0) <= g(3)  or (g(2)  and p(3))  or (g(1)  and p(2)  and p(3))  or (g(0)  and p(1)  and p(2)  and p(3)) ;
pp(1) <= p(4)  and p(5)  and p(6)  and p(7) ;
gg(1) <= g(7)  or (g(6)  and p(7))  or (g(5)  and p(6)  and p(7))  or (g(4)  and p(5)  and p(6)  and p(7)) ;
pp(2) <= p(8)  and p(9)  and p(10)  and p(11) ;
gg(2) <= g(11) or (g(10) and p(11)) or (g(9)  and p(10) and p(11)) or (g(8)  and p(9)  and p(10) and p(11)) ;
pp(3) <= p(12) and p(13) and p(14)  and p(15) ;
gg(3) <= g(15) or (g(14) and p(15)) or (g(13) and p(14) and p(15)) or (g(12) and p(13) and p(14) and p(15)) ;
pp(5) <= p(16) and p(17) and p(18)  and p(19) ;
gg(5) <= g(19) or (g(18) and p(19)) or (g(17) and p(18) and p(19)) or (g(16) and p(17) and p(18) and p(19)) ;
```

```
pp(6) <= p(20) and p(21) and p(22)  and p(23) ;
gg(6) <= g(23) or (g(22) and p(23)) or (g(21) and p(22) and p(23)) or (g(20) and p(21) and p(22) and p(23)) ;
pp(7) <= p(24) and p(25) and p(26)  and p(27) ;
gg(7) <= g(27) or (g(26) and p(27)) or (g(25) and p(26) and p(27)) or (g(24) and p(25) and p(26) and p(27)) ;
pp(8) <= p(28) and p(29) and p(30)  and p(31) ;
gg(8) <= g(31) or (g(30) and p(31)) or (g(29) and p(30) and p(31)) or (g(28) and p(29) and p(30) and p(31)) ;

c(3)  <= gg(0) or (pp(0) and carryin) ;
c(7)  <= gg(1) or (pp(1) and gg(0)) or (pp(0) and pp(1) and carryin) ;
c(11) <= gg(2) or (pp(2) and gg(1)) or (pp(1) and pp(2) and gg(0)) or (pp(0) and pp(1) and pp(2) and carryin) ;
c(19) <= gg(5) or (pp(5) and c(15)) ;
c(23) <= gg(6) or (pp(6) and gg(5)) or (pp(5) and pp(6) and c(15)) ;
c(27) <= gg(7) or (pp(7) and gg(6)) or (pp(6) and pp(7) and gg(5)) or (pp(5) and pp(6) and pp(7) and c(15)) ;

pp(4) <= pp(0) and pp(1) and pp(2) and pp(3) ;
gg(4) <= gg(3) or (gg(2) and pp(3)) or (gg(1) and pp(2) and pp(3)) or (gg(0) and pp(1) and pp(2) and pp(3)) ;
pp(9) <= pp(5) and pp(6) and pp(7) and pp(8) ;
gg(9) <= gg(8) or (gg(7) and pp(8)) or (gg(6) and pp(7) and pp(8)) or (gg(5) and pp(6) and pp(7) and pp(8)) ;

c(15) <= gg(4) or (pp(4) and carryin) ;
c(31) <= gg(9) or (pp(9) and gg(4)) or (pp(4) and pp(9) and carryin) ;

---------------------------------
--  descricao de saida
---------------------------------

        saida(0) <= p(0) xor carryin ;
        saida(1 to 31) <= p(1 to 31) xor c(0 to 30) ;
        carryout <= c(31) ;

end comportamental ;
```

ANEXO II

MULTIPLICADOR

```
--=============================================================
--      Projeto de um MULTIPLICADOR BINARIO de 16 bits
--      Descricao Comportamental de MULTIPLICADOR_BIN.VBE
--      Autor : Romulo P. C. Ferreira
--      Data  : 21/09/92
--      Departamento de Informatica - UFPB - Campus I
--=============================================================

entity multiplicador_bin is

        port (
                num_multiplicando  : in bit_vector(0 to 15) ;   -- numero dividendo de 16 bits
                num_multiplicador  : in bit_vector(0 to 15) ;   -- numero divisor de 16 bits
                inicio             : in bit ;   -- sinal de inicio de divisao
                ckm                : in bit ;   -- clock mestre
                cks                : in bit ;   -- clock escravo
                num_produto        : inout bit_vector(0 to 31) ;   -- numero quociente de 32 bits
                fim                : inout bit ;   -- sinal de fim de divisao
                vdd                : in bit ;   -- alimentacao positiva
                vss                : in bit  -- alimentacao negativa
            ) ;

end multiplicador_bin ;

architecture comportamental of multiplicador_bin is

-----------------------------------------------
--   Sinais do bloco Multiplicador
-----------------------------------------------

signal mux_multiplicador : bit_vector(0 to 15) ;  -- sinal na saida do MUX do Multiplicador
signal mst_multiplicador  : reg_vector(0 to 15) register ;  -- sinal na saida do latch mestre do Multiplicador
signal slv_multiplicador   : reg_vector(0 to 15) register ;  -- sinal na saida do latch escravo do Multiplicador

-----------------------------------------------
--   Sinais do bloco Multiplicando
-----------------------------------------------

signal mst_multiplicando  : reg_vector(0 to 15) register ;  -- sinal na saida do latch mestre do multiplicando
signal slv_multiplicando   : reg_vector(0 to 15) register ;  -- sinal na saida do latch escravo do multiplicando
signal and_16              : bit_vector(0 to 15) ;  -- sinal na saida dos and's
```

```
-------------------------------------
--  Sinais do bloco Somador
-------------------------------------

signal carry      : bit_vector(0 to 16) ;  -- carrys da soma de cada bit
signal soma       : bit_vector(0 to 15) ;  -- sinal na saida do Somador

-------------------------------------
--  Sinais do bloco Produto
-------------------------------------

signal mux_produto      : bit_vector(0 to 31) ;  -- sinal na saida do MUX do Produto
signal nor_32           : bit_vector(0 to 31) ;  -- sinal na saida dos nor's
signal mst_produto      : reg_vector(0 to 31) register ;  -- sinal na saida do latch mestre do Produto
signal slv_produto      : reg_vector(0 to 31) register ;  -- sinal na saida do latch escravo do Produto

-----------------------------------------
--  Sinais do bloco Controlador
-----------------------------------------

signal not_inicio         : bit ;  -- sinal usado para gerar ckm_interno e multiplicando
signal ckm_interno        : bit ;  -- sinal de clock mestre interno
signal cks_interno        : bit ;  -- sinal de clock escravo interno
signal ckm_multiplicador  : bit ;  -- sinal de clock mestre para multiplicador
signal cks_multiplicador  : bit ;  -- sinal de clock escravo para multiplicador
signal ckm_contador       : bit ;  -- sinal de clock mestre para contador
signal cks_contador       : bit ;  -- sinal de clock escravo para contador
signal ckm_inicio         : bit ;  -- sinal de clock mestre para a inicializacao do multiplicador
signal cks_inicio         : bit ;  -- sinal de clock escravo para a inicializacao do multiplicador
signal mst_habilita       : reg_bit register ;  -- sinal na saida do latch mestre do habilita
signal slv_habilita       : reg_bit register ;  -- sinal na saida do latch escravo do habilita

-----------------------------------------
--  Sinais do bloco Finalizador
-----------------------------------------

        signal saida_mst_contador : reg_vector(0 to 5) register ;  -- sinal na saida do latch mestre do contador
        signal saida_slv_contador : reg_vector(0 to 5) register ;  -- sinal na saida do latch escravo do contador


begin

        assert (((vdd xor '1') or (vss xor '0')) = '0')
                report "Alimentacao perdida em MULTIPLICADOR_BIN" severity warning;


-------------------------------------------
--     Descricao do Bloco MULTIPLICADOR
-------------------------------------------

-------------------------------------------------------------------------------------------------------------
--  Descricao do MUX :  quando inicio=0, temos um deslocamento no Multiplicador
--                      quando inicio=1, temos um carregamento paralelo no Multiplicador
--  obs: este bloco e' um registrador de carregamento paralelo e serial (deslocamento)
--          o MUX serve para fazer esta selecao.
-------------------------------------------------------------------------------------------------------------
```

```
with inicio select
        mux_multiplicador(15) <=        num_multiplicador(15) when '0' ,
                                        num_multiplicador(15) when others ;

with inicio select
        mux_multiplicador(0 to 14) <=slv_multiplicador(1 to 15) when '0' ,
                                        num_multiplicador(0 to 14) when others ;

-------------------------------------------
--  Descricao dos Registradores
-------------------------------------------

        reg_mst_multiplicador : block (ckm_multiplicador = '1')
                                begin
                                        mst_multiplicador <= guarded mux_multiplicador ;
                                end block;

        reg_slv_multiplicador : block (cks_multiplicador = '1')
                                begin
                                        slv_multiplicador <= guarded mst_multiplicador ;
                                end block;


--=============================================
--      Descricao do Bloco MULTIPLICANDO
--=============================================

-------------------------------------------------------------------------------
--  O bloco Multiplicando e' um registrador de carregamento paralelo.
-------------------------------------------------------------------------------

        and_16(0)  <= ( slv_multiplicando(0)  and slv_multiplicador(0) ) ;
        and_16(1)  <= ( slv_multiplicando(1)  and slv_multiplicador(0) ) ;
        and_16(2)  <= ( slv_multiplicando(2)  and slv_multiplicador(0) ) ;
        and_16(3)  <= ( slv_multiplicando(3)  and slv_multiplicador(0) ) ;
        and_16(4)  <= ( slv_multiplicando(4)  and slv_multiplicador(0) ) ;
        and_16(5)  <= ( slv_multiplicando(5)  and slv_multiplicador(0) ) ;
        and_16(6)  <= ( slv_multiplicando(6)  and slv_multiplicador(0) ) ;
        and_16(7)  <= ( slv_multiplicando(7)  and slv_multiplicador(0) ) ;
        and_16(8)  <= ( slv_multiplicando(8)  and slv_multiplicador(0) ) ;
        and_16(9)  <= ( slv_multiplicando(9)  and slv_multiplicador(0) ) ;
        and_16(10) <= ( slv_multiplicando(10) and slv_multiplicador(0) ) ;
        and_16(11) <= ( slv_multiplicando(11) and slv_multiplicador(0) ) ;
        and_16(12) <= ( slv_multiplicando(12) and slv_multiplicador(0) ) ;
        and_16(13) <= ( slv_multiplicando(13) and slv_multiplicador(0) ) ;
        and_16(14) <= ( slv_multiplicando(14) and slv_multiplicador(0) ) ;
        and_16(15) <= ( slv_multiplicando(15) and slv_multiplicador(0) ) ;


        reg_mst_multiplicando : block (ckm_inicio = '1')
                                begin
                                        mst_multiplicando <= guarded num_multiplicando ;
                                end block;

        reg_slv_multiplicando : block (cks_inicio = '1')
                                begin
                                        slv_multiplicando <= guarded mst_multiplicando ;
                                end block;
```

```
--=============================
--    Descricao do Bloco SOMADOR
--=============================

-------------------------------------------------------------------
--  Os sinais carry(i) sao os carrys da soma de cada bit.
-------------------------------------------------------------------

        carry(0) <= '0' ;
        carry(1 to 16) <= ((num_produto(15 to 30) and carry(0 to 15)) or
                           (num_produto(15 to 30) and and_16) or
                           (carry(0 to 15) and and_16 )) ;
        soma <= num_produto(15 to 30) xor and_16 xor carry(0 to 15) ;

--=============================
--    Descricao do Bloco PRODUTO
--=============================

-------------------------------------------------------------------------------------------------
--  Este bloco e' um registrador de carregamento paralelo e serial, onde a entrada
--  serial recebe o sinal soma.
--  Descricao do MUX : quando slv_habilita=0, temos um carregamento paralelo.
--                     quando slv_habilita=1, temos um carregamento serial.
-------------------------------------------------------------------------------------------------

        with slv_habilita select
                mux_produto(0 to 14) <= not slv_produto(0 to 14) when '0' ,
                                        not slv_produto(1 to 15) when others ;

        with slv_habilita select
                mux_produto(15 to 30) <= not soma(0 to 15) when '0' ,
                                         not slv_produto(16 to 31) when others ;

        with slv_habilita select
                mux_produto(31) <= not carry(16) when '0' ,
                                   not carry(16) when others ;

-------------------------------------------------------------------
--  Descricao do estagio de RESET.
--  obs: Quando inicio=1 => nor_32=0 => produto=0
-------------------------------------------------------------------

        nor_32(0)  <= not ( mux_produto(0)  or inicio ) ;
        nor_32(1)  <= not ( mux_produto(1)  or inicio ) ;
        nor_32(2)  <= not ( mux_produto(2)  or inicio ) ;
        nor_32(3)  <= not ( mux_produto(3)  or inicio ) ;
        nor_32(4)  <= not ( mux_produto(4)  or inicio ) ;
        nor_32(5)  <= not ( mux_produto(5)  or inicio ) ;
        nor_32(6)  <= not ( mux_produto(6)  or inicio ) ;
        nor_32(7)  <= not ( mux_produto(7)  or inicio ) ;
        nor_32(8)  <= not ( mux_produto(8)  or inicio ) ;
        nor_32(9)  <= not ( mux_produto(9)  or inicio ) ;
        nor_32(10) <= not ( mux_produto(10) or inicio ) ;
        nor_32(11) <= not ( mux_produto(11) or inicio ) ;
        nor_32(12) <= not ( mux_produto(12) or inicio ) ;
        nor_32(13) <= not ( mux_produto(13) or inicio ) ;
        nor_32(14) <= not ( mux_produto(14) or inicio ) ;
        nor_32(15) <= not ( mux_produto(15) or inicio ) ;
```

```
        nor_32(16) <= not ( mux_produto(16) or inicio ) ;
        nor_32(17) <= not ( mux_produto(17) or inicio ) ;
        nor_32(18) <= not ( mux_produto(18) or inicio ) ;
        nor_32(19) <= not ( mux_produto(19) or inicio ) ;
        nor_32(20) <= not ( mux_produto(20) or inicio ) ;
        nor_32(21) <= not ( mux_produto(21) or inicio ) ;
        nor_32(22) <= not ( mux_produto(22) or inicio ) ;
        nor_32(23) <= not ( mux_produto(23) or inicio ) ;
        nor_32(24) <= not ( mux_produto(24) or inicio ) ;
        nor_32(25) <= not ( mux_produto(25) or inicio ) ;
        nor_32(26) <= not ( mux_produto(26) or inicio ) ;
        nor_32(27) <= not ( mux_produto(27) or inicio ) ;
        nor_32(28) <= not ( mux_produto(28) or inicio ) ;
        nor_32(29) <= not ( mux_produto(29) or inicio ) ;
        nor_32(30) <= not ( mux_produto(30) or inicio ) ;
        nor_32(31) <= not ( mux_produto(31) or inicio ) ;

-------------------------------------------
--  Descricao dos Registradores
-------------------------------------------

        reg_mst_produto : block (ckm_interno = '1')
                          begin
                                mst_produto <= guarded nor_32 ;
                          end block;

        reg_slv_produto : block (cks_interno = '1')
                          begin
                                slv_produto <= guarded mst_produto ;
                          end block;

        num_produto <= slv_produto ;

--===========================================
--    Descricao do Bloco CONTROLADOR
--===========================================

-----------------------------------
--  Descricao dos clocks
-----------------------------------

          not_inicio          <= not inicio ;
          ckm_interno         <= ckm and not_inicio and fim ;
          cks_interno         <= cks and cks and cks ;
          ckm_inicio          <= inicio and ckm ;
          cks_inicio          <= cks and cks ;
          ckm_multiplicador   <= ckm_inicio or (slv_habilita and ckm_interno) ;
          cks_multiplicador   <= (cks_interno and cks_interno) or (cks_interno and cks_interno) ;
          ckm_contador        <= ckm_inicio or ckm_interno ;
          cks_contador        <= cks_interno or cks_interno ;

---------------------------------------
--  Descricao do habilitador
---------------------------------------
        reg_mst_habilita : block (ckm_contador = '1')
                           begin
                                 mst_habilita <= guarded not (inicio or slv_habilita) ;
                           end block;
```

```
        reg_slv_habilita : block (cks_contador = '1')
                           begin
                                slv_habilita <= guarded mst_habilita ;
                           end block;

--=============================================
--    Descricao do Bloco FINALIZADOR
--=============================================

-------------------------------------
--  Descricao do contador
-------------------------------------

        reg_mst_contador : block (ckm_contador = '1')
                    begin
                          saida_mst_contador(0) <= guarded not (inicio or saida_slv_contador(0)) ;
                          saida_mst_contador(1) <= guarded not (inicio or not (saida_slv_contador(0) xor
                                                                saida_slv_contador(1))) ;
                          saida_mst_contador(2) <= guarded not (inicio or not (saida_slv_contador(2) xor
                                                               (saida_slv_contador(0) and
                                                                saida_slv_contador(1))));
                          saida_mst_contador(3) <= guarded not (inicio or not (saida_slv_contador(3) xor
                                                               (saida_slv_contador(0) and
                                                                saida_slv_contador(1) and
                                                                saida_slv_contador(2))));
                          saida_mst_contador(4) <= guarded not (inicio or not (saida_slv_contador(4) xor
                                                               (saida_slv_contador(0) and
                                                                saida_slv_contador(1) and
                                                                saida_slv_contador(2) and
                                                                saida_slv_contador(3))));
                          saida_mst_contador(5) <= guarded not (inicio or not(saida_slv_contador(5) xor
                                                               (saida_slv_contador(0) and
                                                                saida_slv_contador(1) and
                                                                saida_slv_contador(2) and
                                                                saida_slv_contador(3) and
                                                                saida_slv_contador(4))));
                    end block;

        reg_slv_contador : block (cks_contador = '1')
                           begin
                                saida_slv_contador <= guarded saida_mst_contador ;
                           end block;

-------------------------------------------------
--  Descricao da logica de finalizacao
-------------------------------------------------

        fim <= not (saida_slv_contador(0) and
                    saida_slv_contador(1) and
                    saida_slv_contador(2) and
                    saida_slv_contador(3) and
                    saida_slv_contador(4)) ;

end comportamental ;
```

ANEXO III

DIVISOR

```
--=============================================================
--      Projeto de um DIVISOR BINARIO de 8 bits
--      Descricao Comportamental de DIVISOR_BIN.VBE
--      Autor : Romulo P. C. Ferreira
--      Data  : 10/08/92
--      Departamento de Informatica - UFPB - Campus I
--=============================================================

entity divisor_bin is

        port (
                num_dividendo   : in bit_vector(0 to 31) ;  -- numero dividendo de 8 bits
                num_divisor     : in bit_vector(0 to 10) ;  -- numero divisor de 4 bits
                inicio          : in bit ;  -- sinal de inicio de divisao
                ckm             : in bit ;  -- clock mestre
                cks             : in bit ;  -- clock escravo
                num_quociente   : out bit_vector(0 to 31) ;  -- numero quociente de 8 bits
                fim             : inout bit ;  -- sinal de fim de divisao
                vdd             : in bit ;  -- alimentacao positiva
                vss             : in bit  -- alimentacao negativa
            ) ;

end divisor_bin ;


architecture comportamental of divisor_bin is

-----------------------------------------
--  Sinais do bloco Dividendo
-----------------------------------------

signal mux_dividendo        : bit_vector(0 to 31)       ; -- sinal na saida do MUX do Dividendo
signal saida_mst_dividendo  : reg_vector(0 to 31) register ; -- sinal na saida do latch mestre do dividendo
signal saida_slv_dividendo  : reg_vector(0 to 31) register ; -- sinal na saida do latch escravo do dividendo

-------------------------------------
--  Sinais do bloco Divisor
-------------------------------------

        signal saida_mst_divisor  : reg_vector(0 to 10) register ; -- sinal na saida do latch mestre do divisor
        signal saida_slv_divisor  : reg_vector(0 to 10) register ; -- sinal na saida do latch escravo do divisor
```

```
-------------------------------------
--  Sinais do bloco Resto
-------------------------------------

        signal mux_resto        : bit_vector(0 to 11) ;  -- sinal na saida do MUX do Resto
        signal saida_nor        : bit_vector(0 to 11) ;  -- sinal de saida do estagio de RESET
        signal saida_mst_resto  : reg_vector(0 to 11) register ;  -- sinal na saida do latch mestre do resto
        signal saida_slv_resto  : reg_vector(0 to 11) register ;  -- sinal na saida do latch escravo do resto

-------------------------------------
--  Sinais do bloco Somador
-------------------------------------

          signal carry          : bit_vector(0 to 12) ;  -- carrys da soma de cada bit
          signal complemento_2  : bit_vector(0 to 11) ;  -- complemento de dois do numero divisor
          signal soma           : bit_vector(0 to 11) ;  -- sinal na saida do Somador

-----------------------------------------
--  Sinais do bloco Quociente
-----------------------------------------

      signal saida_mst_quociente  : reg_vector(0 to 31) register ;  -- sinal na saida do latch mestre do quociente
      signal saida_slv_quociente  : reg_vector(0 to 31) register ;  -- sinal na saida do latch escravo do quociente

-----------------------------------------
--  Sinais do bloco Controlador
-----------------------------------------

        signal not_inicio         : bit ;  -- sinal usado para gerar ckm_interno
        signal not_carryout       : bit ;  -- sinal usado para gerar ckm_trio
        signal ckm_interno        : bit ;  -- sinal de clock mestre habilitador do divisor
        signal cks_interno        : bit ;  -- sinal de clock escravo habilitador do divisor
        signal ckm_dividendo      : bit ;  -- sinal de clock mestre para dividendo
        signal cks_dividendo      : bit ;  -- sinal de clock escravo para dividendo
        signal ckm_quociente      : bit ;  -- sinal de clock mestre para quociente
        signal cks_quociente      : bit ;  -- sinal de clock escravo para quociente
        signal ckm_contador       : bit ;  -- sinal de clock mestre para contador
        signal cks_contador       : bit ;  -- sinal de clock escravo para contador
        signal ckm_resto          : bit ;  -- sinal de clock mestre para o bloco resto
        signal cks_resto          : bit ;  -- sinal de clock escravo para o bloco resto
        signal ckm_inicio         : bit ;  -- sinal de clock mestre para a inicializacao do divisor
        signal cks_inicio         : bit ;  -- sinal de clock escravo para a inicializacao do divisor
        signal saida_mst_habilita : reg_bit register ;  -- sinal na saida do latch mestre do habilita
        signal saida_slv_habilita : reg_bit register ;  -- sinal na saida do latch escravo do habilita

-----------------------------------------
--  Sinais do bloco Finalizador
-----------------------------------------

      signal saida_mst_contador   : reg_vector(0 to 5) register ;  -- sinal na saida do latch mestre do contador
      signal saida_slv_contador   : reg_vector(0 to 5) register ;  -- sinal na saida do latch escravo do contador


begin

        assert (((vdd xor '1') or (vss xor '0')) = '0')
                report "Alimentacao perdida em DIVISOR_BIN" severity warning;
```

```
--=============================================================
--      Descricao do Bloco DIVIDENDO
--=============================================================

-------------------------------------------------------------------------------------------------------------
--  Descricao do MUX : quando inicio=0, temos um deslocamento no dividendo
--                     quando inicio=1, temos um carregamento paralelo no dividendo
--  obs: este bloco e' um registrador de carregamento paralelo e serial (deslocamento)
--       o MUX serve para fazer esta selecao.
-------------------------------------------------------------------------------------------------------------

        with inicio select
                mux_dividendo(0) <= num_dividendo(0) when '0' ,
                                    num_dividendo(0) when others ;

        with inicio select
                mux_dividendo(1 to 31) <= saida_slv_dividendo(0 to 30) when '0' ,
                                          num_dividendo(1 to 31) when others ;

-----------------------------------------
--  Descricao dos Registradores
-----------------------------------------

        reg_mst_dividendo : block (ckm_dividendo = '1')
                            begin
                                saida_mst_dividendo <= guarded mux_dividendo ;
                            end block;

        reg_slv_dividendo : block (cks_dividendo = '1')
                            begin
                                saida_slv_dividendo <= guarded saida_mst_dividendo ;
                            end block;


--=================================
--      Descricao do Bloco DIVISOR
--=================================

-------------------------------------------------------------------------------------------------------------
--  O bloco divisor e' um registrador de carregamento paralelo.
--  obs: O sinal saida_slv_divisor e' invertido para fornecer ao somador, o complemento de um.
-------------------------------------------------------------------------------------------------------------

        reg_mst_divisor : block (ckm_inicio = '1')
                          begin
                              saida_mst_divisor <= guarded num_divisor ;
                          end block;

        reg_slv_divisor : block (cks_inicio = '1')
                          begin
                              saida_slv_divisor <= guarded not saida_mst_divisor ;
                          end block;


--=================================
--      Descricao do Bloco RESTO
--=================================
```

```
-- O bloco resto e' um registrador de carregamento paralelo e serial (deslocamento),
-- assim, o mux serve para fazer esta selecao. Este bloco ainda inclui o RESET.
-- Descricao do MUX : quando carryout=0, temos um deslocamento no resto.
--                    quando carryout=1, temos um carregamento paralelo no resto.
-- carryout = carry(5)
-----------------------------------------------------------------------------------------------------------------

        with carry(12) select
                mux_resto(0) <= not saida_slv_dividendo(31) when '0' ,
                                not soma(0) when others ;

        with carry(12) select
                mux_resto(1 to 11) <= not saida_slv_resto(0 to 10) when '0' ,
                                      not soma(1 to 11) when others ;

-----------------------------------------------------------------------
-- Descricao do estagio de RESET.
-- obs: Quando inicio=1 => saida_nor=0 => resto=0
-----------------------------------------------------------------------

        saida_nor(0)  <= not ( mux_resto(0)  or inicio ) ;
        saida_nor(1)  <= not ( mux_resto(1)  or inicio ) ;
        saida_nor(2)  <= not ( mux_resto(2)  or inicio ) ;
        saida_nor(3)  <= not ( mux_resto(3)  or inicio ) ;
        saida_nor(4)  <= not ( mux_resto(4)  or inicio ) ;
        saida_nor(5)  <= not ( mux_resto(5)  or inicio ) ;
        saida_nor(6)  <= not ( mux_resto(6)  or inicio ) ;
        saida_nor(7)  <= not ( mux_resto(7)  or inicio ) ;
        saida_nor(8)  <= not ( mux_resto(8)  or inicio ) ;
        saida_nor(9)  <= not ( mux_resto(9)  or inicio ) ;
        saida_nor(10) <= not ( mux_resto(10) or inicio ) ;
        saida_nor(11) <= not ( mux_resto(11) or inicio ) ;

-----------------------------------------
-- Descricao dos Registradores
-----------------------------------------

        reg_mst_resto : block (ckm_resto = '1')
                        begin
                                saida_mst_resto <= guarded saida_nor ;
                        end block;

        reg_slv_resto : block (cks_resto = '1')
                        begin
                                saida_slv_resto <= guarded  saida_mst_resto ;
                        end block;

--=====================================
--    Descricao do Bloco SOMADOR
--=====================================

-----------------------------------------------------------------------------------------------------------------
-- Os sinais carry(i) sao os carrys da soma de cada bit. Carry(o)=1 e' o carryin para o
-- complemento de dois do numero divisor.
-- No sinal complemento_2 temos complemento_2(11)=1 porque o MSB do numero divisor e' '0',
-- assim, o seu complemento sera' '1'.
-----------------------------------------------------------------------------------------------------------------
```

```
        carry(0) <= '1' ;
        complemento_2(0 to 10) <= saida_slv_divisor ;
        complemento_2(11) <= '1' ;
        carry(1 to 12) <= ((complemento_2 and carry(0 to 11)) or
                           (complemento_2 and saida_slv_resto) or
                           (carry(0 to 11) and saida_slv_resto )) ;
        soma <= complemento_2 xor saida_slv_resto xor carry(0 to 11) ;


--=============================================
--      Descricao do Bloco QUOCIENTE
--=============================================

-------------------------------------------------------------------------------------------
--  Este bloco e' um registrador de carregamento serial, onde a entrada
--  serial recebe o sinal carryout ( carry(5) )
-------------------------------------------------------------------------------------------

        reg_mst_quociente : block (ckm_quociente = '1')
                            begin
                                saida_mst_quociente(0)      <= guarded carry(12) ;
                                saida_mst_quociente(1 to 31) <= guarded saida_slv_quociente(0 to 30) ;
                            end block;

        reg_slv_quociente : block (cks_quociente = '1')
                            begin
                                saida_slv_quociente <= guarded saida_mst_quociente ;
                            end block;

        num_quociente <= saida_slv_quociente ;


--=============================================
--      Descricao do Bloco CONTROLADOR
--=============================================

-----------------------------------
--  Descricao dos clocks
-----------------------------------

        not_inicio      <= not inicio ;
        not_carryout    <= not carry(12) ;
        ckm_interno     <= ckm and not_inicio and fim ;
        cks_interno     <= cks and cks and cks ;
        ckm_inicio      <= inicio and ckm ;
        cks_inicio      <= cks and cks ;
        ckm_dividendo   <= ckm_inicio or (not_carryout and ckm_interno) ;
        cks_dividendo   <= (cks_interno and cks_interno) or (cks_interno and cks_interno) ;
        ckm_quociente   <= saida_slv_habilita and ckm_interno ;
        cks_quociente   <= cks_interno and cks_interno ;
        ckm_contador    <= ckm_inicio or ckm_quociente ;
        cks_contador    <= cks_quociente or cks_quociente ;
        ckm_resto       <= ckm_inicio or ckm_interno ;
        cks_resto       <= cks_interno or cks_interno ;

---------------------------------------
--  Descricao do habilitador
---------------------------------------
```

```
        reg_mst_habilita : block (ckm_resto = '1')
                           begin
                                saida_mst_habilita <= guarded not (inicio or carry(12)) ;
                           end block;

        reg_slv_habilita : block (cks_resto = '1')
                           begin
                                saida_slv_habilita <= guarded saida_mst_habilita ;
                           end block;

--=====================================
--    Descricao do Bloco FINALIZADOR
--=====================================

-------------------------------------
--  Descricao do contador
-------------------------------------

        reg_mst_contador : block (ckm_contador = '1')
                    begin
                         saida_mst_contador(0) <= guarded not (inicio or saida_slv_contador(0)) ;
                         saida_mst_contador(1) <= guarded not (inicio or not (saida_slv_contador(0) xor
                                                                saida_slv_contador(1))) ;
                         saida_mst_contador(2) <= guarded not (inicio or not (saida_slv_contador(2) xor
                                                        (saida_slv_contador(0) and
                                                         saida_slv_contador(1))));
                         saida_mst_contador(3) <= guarded not (inicio or not (saida_slv_contador(3) xor
                                                        (saida_slv_contador(0) and
                                                         saida_slv_contador(1) and
                                                         saida_slv_contador(2))));
                         saida_mst_contador(4) <= guarded not (inicio or not (saida_slv_contador(4) xor
                                                        (saida_slv_contador(0) and
                                                         saida_slv_contador(1) and
                                                         saida_slv_contador(2) and
                                                         saida_slv_contador(3))));
                         saida_mst_contador(5) <= guarded not (inicio or not(saida_slv_contador(5) xor
                                                        (saida_slv_contador(0) and
                                                         saida_slv_contador(1) and
                                                         saida_slv_contador(2) and
                                                         saida_slv_contador(3) and
                                                         saida_slv_contador(4))));
                    end block;

        reg_slv_contador : block (cks_contador = '1')
                           begin
                                saida_slv_contador <= guarded saida_mst_contador ;
                           end block;

-------------------------------------------------
--  Descricao da logica de finalizacao
-------------------------------------------------

        fim <= not saida_slv_contador(5) ;


end comportamental ;
```

ANEXO IV

UAB

```
--=============================================================
--      Projeto de uma UNIDADE ARITMETICA BASICA de 32 bits
--      Descricao Comportamental de UAB_CORE.VBE
--      Autor : Romulo P. C. Ferreira
--      Data  : 05/09/92
--      Departamento de Informatica - UFPB - Campus I
--=============================================================

entity uab_core is

        port (
                entrada  : in  bit_vector(0 to 15) ;
                sel_out  : in  bit ;
                o0       : in  bit ;
                o1       : in  bit ;
                load     : in  bit ;
                reset    : in  bit ;
                opera    : in  bit ;
                ckm      : in  bit ;
                cks      : in  bit ;
                saida    : out bit_vector(0 to 15) ;
                fim      : out bit ;
                carryout : out bit ;
                vdd      : in  bit ;
                vss      : in  bit
            ) ;

end uab_core ;

architecture comportamental of uab_core is

---------------------------------
--  Sinais do bloco RP
---------------------------------

        signal and16_rp    : bit_vector(0 to 31) ;
        signal out_mst_rp  : reg_vector(0 to 31) register ;
        signal out_slv_rp  : reg_vector(0 to 31) register ;
        signal ckm_rp      : bit ;
        signal cks_rp      : bit ;
```

```
-------------------------------------
--  Sinais do bloco RSPED
-------------------------------------

        signal ckm_rsped01      : bit ;
        signal cks_rsped01      : bit ;
        signal c1_rsped01       : bit ;
        signal c2_rsped01       : bit ;
        signal and16_rsped01    : bit_vector(0 to 31) ;
        signal out_mux_rsped01  : bit_vector(0 to 31) ;
        signal out_mst_rsped01  : reg_vector(0 to 31) register ;
        signal out_slv_rsped01  : reg_vector(0 to 31) register ;

        signal ckm_rsped23      : bit ;
        signal cks_rsped23      : bit ;
        signal and16_rsped23    : bit_vector(0 to 31) ;
        signal out_mux_rsped23  : bit_vector(0 to 31) ;
        signal out_mst_rsped23  : reg_vector(0 to 31) register ;
        signal out_slv_rsped23  : reg_vector(0 to 31) register ;

        signal ckm_rsped45      : bit ;
        signal cks_rsped45      : bit ;
        signal c1_rsped45       : bit ;
        signal c2_rsped45       : bit ;
        signal and16_rsped45    : bit_vector(0 to 31) ;
        signal out_mux_rsped45  : bit_vector(0 to 31) ;
        signal out_mst_rsped45  : reg_vector(0 to 31) register ;
        signal out_slv_rsped45  : reg_vector(0 to 31) register ;

-----------------------------------------
--  Sinais do bloco COMPLAND
-----------------------------------------

        signal nand16_compland  : bit_vector(0 to 31) ;
        signal out_compland     : bit_vector(0 to 31) ;

-----------------------------------------
--  Sinais do bloco SOMADOR
-----------------------------------------

        signal carry    : bit_vector(0 to 32) ;
        signal soma     : bit_vector(0 to 31) ;

-----------------------------------------
--  Sinais do bloco CTRL_CLOCK
-----------------------------------------

        signal ckm_cont6        : bit ;
        signal cks_cont6        : bit ;
        signal h_interno        : bit ;
        signal dmfim            : bit ;
        signal ckm_o            : bit ;
        signal cks_o            : bit ;
        signal ckm_cont2        : bit ;
        signal cks_cont2        : bit ;
        signal ckm_mfim         : bit ;
        signal cks_mfim         : bit ;
        signal ckm_dfim         : bit ;
```

```
        signal cks_dfim         : bit ;
        signal h_load           : bit ;
        signal out_mst_cont     : reg_vector(0 to 1) register ;
        signal out_slv_cont     : reg_vector(0 to 1) register ;
        signal out_mstm         : reg_bit register ;
        signal out_slvm         : reg_bit register ;
        signal out_mstd         : reg_bit register ;
        signal out_slvd         : reg_bit register ;
        signal out_mst_o0       : reg_bit register ;
        signal out_slv_o0       : reg_bit register ;
        signal out_mst_o1       : reg_bit register ;
        signal out_slv_o1       : reg_bit register ;
        signal out_muxd         : bit ;
        signal out_muxm         : bit ;
        signal mhabilita        : bit ;
        signal final            : bit ;

-----------------------------------------------
--  Sinais do bloco FINALIZADOR
-----------------------------------------------

        signal out_mst_cont6    : reg_vector(0 to 5) register ;
        signal out_slv_cont6    : reg_vector(0 to 5) register ;
        signal dfim             : bit ;
        signal mfim             : bit ;

-----------------------------------------------
--  Sinais do MUX do SOMADOR
-----------------------------------------------

        signal c1_soma          : bit ;
        signal c2_soma          : bit ;
        signal out_mux_soma     : bit_vector(0 to 31) ;

-----------------------------------------------
--  Sinais do MUX de ENTRADA
-----------------------------------------------

        signal out_mux_entrada0 : bit_vector(0 to 15) ;
        signal out_mux_entrada1 : bit_vector(0 to 15) ;
        signal out_mux_entrada2 : bit_vector(0 to 15) ;
        signal out_mux_entrada3 : bit_vector(0 to 15) ;

-----------------------------------------
--  Sinal do MUX de SAIDA
-----------------------------------------

        signal out_mux_saida    : bit_vector(0 to 31) ;
        signal not_reset        : bit_vector(0 to 31) ;
        signal aux_cand         : bit_vector(0 to 31) ;


begin

        assert (((vdd xor '1') or (vss xor '0')) = '0')
                report "Alimentacao perdida em UAB_CORE" severity warning;
```

```
        not_reset(0) <= not reset ;
        not_reset(1 to 31) <= not_reset(0 to 30) ;


--=================================
--      Descricao do Bloco RSPED01
--=================================


-------------------------------------
--  Descricao do MUX 3:1
-------------------------------------

        with (c1_rsped01 & c2_rsped01) select
                out_mux_rsped01(0) <= '0'                       when B"00" ,
                                      out_mux_entrada0(0) when B"01" ,
                                      out_slv_rsped01(1)  when others ;

        with (c1_rsped01 & c2_rsped01) select
                out_mux_rsped01(1 to 15) <= out_slv_rsped01(0 to 14)  when B"00" ,
                                            out_mux_entrada0(1 to 15) when B"01" ,
                                            out_slv_rsped01(2 to 16)  when others ;

        with (c1_rsped01 & c2_rsped01) select
                out_mux_rsped01(16 to 30) <= out_slv_rsped01(15 to 29) when B"00" ,
                                             out_mux_entrada1(0 to 14) when B"01" ,
                                             out_slv_rsped01(17 to 31) when others ;

        with (c1_rsped01 & c2_rsped01) select
                out_mux_rsped01(31) <= out_slv_rsped01(30)  when B"00" ,
                                       out_mux_entrada1(15) when B"01" ,
                                       '0'                          when others ;


        c1_rsped01 <= not (out_slv_o1 or load)  ;
        c2_rsped01 <= load ;

-----------------------------------------------------
--  Descricao dos ANDs de 2 entradas
-----------------------------------------------------

        and16_rsped01 <= out_mux_rsped01 and not_reset ;

---------------------------------------
--  Descricao dos Registros
---------------------------------------

        reg_mst_rsped01 : block (ckm_rsped01 = '1')
                          begin
                                  out_mst_rsped01 <= guarded and16_rsped01 ;
                          end block ;

        reg_slv_rsped01 : block (cks_rsped01 = '1')
                          begin
                                  out_slv_rsped01 <= guarded out_mst_rsped01 ;
                          end block ;
```

```
--=================================
--    Descricao do Bloco RSPED23
--=================================

-------------------------------------
--  Descricao do MUX 3:1
-------------------------------------

        with carry(32) select
                out_mux_rsped23(0) <= out_slv_rsped01(31) when '0' ,
                                      soma(0)     when others ;

        with carry(32) select
                 out_mux_rsped23(1 to 31) <= out_slv_rsped23(0 to 30) when '0' ,
                                             soma(1 to 31)                when others ;

-----------------------------------------------------
--  Descricao dos ANDs de 2 entradas
-----------------------------------------------------

        and16_rsped23 <= out_mux_rsped23 and not_reset ;

---------------------------------------
--  Descricao dos Registros
---------------------------------------

        reg_mst_rsped23 : block (ckm_rsped23 = '1')
                          begin
                                  out_mst_rsped23 <= guarded and16_rsped23 ;
                          end block ;

        reg_slv_rsped23 : block (cks_rsped23 = '1')
                           begin
                                  out_slv_rsped23 <= guarded out_mst_rsped23 ;
                           end block ;


--=================================
--    Descricao do Bloco RSPED45
--=================================
-------------------------------------
--  Descricao do MUX 3:1
-------------------------------------

        with (c1_rsped45 & c2_rsped45) select
                out_mux_rsped45(0) <= carry(32)              when B"00" ,
                                      out_slv_rsped45(0) when B"01" ,
                                      out_slv_rsped45(1) when others ;

        with (c1_rsped45 & c2_rsped45) select
                out_mux_rsped45(1 to 14) <= out_slv_rsped45(0 to 13) when B"00" ,
                                            out_slv_rsped45(1 to 14) when B"01" ,
                                            out_slv_rsped45(2 to 15) when others ;

        with (c1_rsped45 & c2_rsped45) select
                 out_mux_rsped45(15 to 30) <= out_slv_rsped45(14 to 29) when B"00" ,
                                              soma(0 to 15)                 when B"01" ,
                                              out_slv_rsped45(16 to 31) when others ;
```

```
        with (c1_rsped45 & c2_rsped45) select
                out_mux_rsped45(31) <= out_slv_rsped45(30) when B"00" ,
                                       soma(16)            when B"01" ,
                                       '0'                 when others ;

        c1_rsped45 <= (mhabilita and (not out_slv_o1)) ;
        c2_rsped45 <= ((not mhabilita) and (not out_slv_o1)) ;

-------------------------------------------------
--  Descricao dos ANDs de 2 entradas
-------------------------------------------------

        and16_rsped45 <= out_mux_rsped45 and not_reset ;

---------------------------------------
--  Descricao dos Registros
---------------------------------------

        reg_mst_rsped45 : block (ckm_rsped45 = '1')
                          begin
                                  out_mst_rsped45 <= guarded and16_rsped45 ;
                          end block ;

        reg_slv_rsped45 : block (cks_rsped45 = '1')
                          begin
                                  out_slv_rsped45 <= guarded out_mst_rsped45 ;
                          end block ;


--=========================
--     Descricao do Bloco RP
--=========================


        and16_rp(0 to 15)  <= out_mux_entrada2(0 to 15) and not_reset(0 to 15) ;
        and16_rp(16 to 31) <= out_mux_entrada3(0 to 15) and not_reset(0 to 15) ;

---------------------------------------
--  Descricao dos Registros
---------------------------------------

        reg_mst_rp : block (ckm_rp = '1')
                     begin
                          out_mst_rp <= guarded and16_rp ;
                     end block ;

        reg_slv_rp : block (cks_rp = '1')
                     begin
                          out_slv_rp <= guarded out_mst_rp ;
                     end block ;


--=====================================
--     Descricao do Bloco COMPLAND
--=====================================

        aux_cand(0) <= (out_slv_o0 or out_slv_rsped01(0)) ;
```

```
        aux_cand(1 to 31) <= aux_cand(0 to 30) ;

        nand16_compland <= not (out_slv_rp and aux_cand ) ;

-------------------------------------
--  Descricao do MUX 2:1
-------------------------------------

        with out_slv_o1 select
                out_compland(0 to 31) <= not out_slv_rp      when '1' ,
                                         not nand16_compland when others ;


--=====================================
--      Descricao do Bloco SOMADOR
--=====================================

        carry(0) <= out_slv_o1 ;
        carry(1 to 32) <= ((out_compland and carry(0 to 31)) or
                           (out_compland and out_mux_soma) or
                           (carry(0 to 31) and out_mux_soma )) ;
        soma <= out_compland xor out_mux_soma xor carry(0 to 31) ;
        carryout <= not carry(32) ;


--=====================================
--      Descricao do Bloco CTRL_CLOCK
--=====================================

-------------------------------------------
--  Descricao dos habilitadores
-------------------------------------------

        h_interno <= (final and (not opera)) ;
        h_load    <= (load and (not final)) ;

-------------------------------------------
--  Descricao dos Registradores
-------------------------------------------

        reg_msto : block (ckm_o = '1')
                   begin
                     out_mst_o0 <= guarded (o0 or reset) ; -- reset forca fim ir p/ 0
                     out_mst_o1 <= guarded o1 ;
                   end block;

        reg_slvo : block (cks_o = '1')
                   begin
                     out_slv_o0 <= guarded out_mst_o0 ;
                     out_slv_o1 <= guarded out_mst_o1 ;
                   end block;

        reg_mstm : block (ckm_mfim = '1')
                   begin
                     out_mstm <= guarded (not (mhabilita or opera)) ;
                   end block;

        reg_slvm : block (cks_mfim = '1')
```

```
                begin
                  out_slvm <= guarded out_mstm ;
                end block;

        reg_mstd : block (ckm_rsped23 = '1')
                begin
                  out_mstd <= guarded not (carry(32) or reset) ;
                end block;

        reg_slvd : block (cks_rsped23 = '1')
                begin
                  out_slvd <= guarded out_mstd ;
                end block;

        mhabilita <= out_slvm ;

-------------------------------------
--  Descricao dos MUX 2:1
-------------------------------------

        with out_slv_o1 select
                dmfim <=  dfim when '1' ,
                          mfim when others ;

        with out_slv_o1 select
                out_muxd <= (out_slvd and h_interno) when '1' ,
                             h_interno               when others ;

        with out_slv_o1 select
                out_muxm <= ((not carry(32)) and h_interno) when '1' ,
                            (h_interno and mhabilita)      when others ;

        final <= (dmfim and (not out_slv_o0)) ;
        fim   <=  not final ;

------------------------------
--  Clock's de Saida
------------------------------

        ckm_rsped01   <= ckm and (reset or h_load or out_muxm) ;
        cks_rsped01   <= cks and cks ;
        ckm_rp        <= ckm and (reset or h_load) ;
        cks_rp        <= cks and cks ;
        ckm_rsped45   <= ckm and (reset or out_muxd) ;
        cks_rsped45   <= cks and cks ;
        ckm_rsped23   <= ckm and (reset or h_interno) ;
        cks_rsped23   <= cks and cks ;
        ckm_mfim      <= ckm and (opera or h_interno) ;
        cks_mfim      <= cks and cks ;
        ckm_o         <= ckm and (opera or reset) ;
        cks_o         <= cks and cks ;
        ckm_dfim      <= ckm and (opera or (h_interno and out_slvd)) ;
        cks_dfim      <= cks and cks ;

        with out_slv_o1 select
                ckm_cont6 <=    ckm and (opera or (final and out_slvd)) when '1' ,
                                ckm and (opera or final)  when others ;
        cks_cont6 <= cks and cks ;
```

```
        ckm_cont2 <= ckm and (load or reset) ;
        cks_cont2 <= (cks and (not load)) ;

-------------------------------------------------
--  Descricao do contador de dois bits
-------------------------------------------------

        reg_mst_cont :  block (ckm_cont2 = '1')
                        begin
                           out_mst_cont(0) <= guarded not (reset or out_slv_cont(0)) ;
                           out_mst_cont(1) <= guarded not (reset or not (out_slv_cont(0) xor
                                                                         out_slv_cont(1))) ;
                        end block ;

        reg_slv_cont :  block (cks_cont2 = '1')
                        begin
                           out_slv_cont <= guarded out_mst_cont ;
                        end block;


--=====================================
--      Descricao do Bloco FINALIZADOR
--=====================================

-------------------------------------
--  Descricao do contador
-------------------------------------

        reg_mst_cont6 : block (ckm_cont6 = '1')
                        begin
                              out_mst_cont6(0) <= guarded not (opera or out_slv_cont6(0)) ;
                              out_mst_cont6(1) <= guarded not (opera or not (out_slv_cont6(0) xor
                                                                             out_slv_cont6(1))) ;
                              out_mst_cont6(2) <= guarded not (opera or not (out_slv_cont6(2) xor
                                                                             (out_slv_cont6(0) and
                                                                              out_slv_cont6(1))));
                              out_mst_cont6(3) <= guarded not (opera or not (out_slv_cont6(3) xor
                                                                             (out_slv_cont6(0) and
                                                                              out_slv_cont6(1) and
                                                                              out_slv_cont6(2))));
                              out_mst_cont6(4) <= guarded not (opera or not (out_slv_cont6(4) xor
                                                                             (out_slv_cont6(0) and
                                                                              out_slv_cont6(1) and
                                                                              out_slv_cont6(2) and
                                                                              out_slv_cont6(3))));
                              out_mst_cont6(5) <= guarded not (opera or not (out_slv_cont6(5) xor
                                                                             (out_slv_cont6(0) and
                                                                              out_slv_cont6(1) and
                                                                              out_slv_cont6(2) and
                                                                              out_slv_cont6(3) and
                                                                              out_slv_cont6(4))));
                        end block;

        reg_slv_cont6 : block (cks_cont6 = '1')
                        begin
                              out_slv_cont6 <= guarded out_mst_cont6 ;
                        end block;
```

```
-------------------------------------------------
--  Descricao da logica de finalizacao
-------------------------------------------------

        dfim <= not out_slv_cont6(5) ;
        mfim <= not (out_slv_cont6(0) and out_slv_cont6(1) and
                        out_slv_cont6(2) and out_slv_cont6(3) and out_slv_cont6(4));


--=====================================
--      Descricao do MUX do SOMADOR
--=====================================

        with (c1_soma & c2_soma) select
                out_mux_soma(0 to 15) <= out_slv_rsped45(15 to 30) when B"00" ,
                                         out_slv_rsped01(0 to 15)  when B"01" ,
                                         out_slv_rsped23(0 to 15)  when others ;

        with (c1_soma & c2_soma) select
                 out_mux_soma(16 to 31) <= X"0000"      when B"00" ,
                                           out_slv_rsped01(16 to 31) when B"01" ,
                                           out_slv_rsped23(16 to 31) when others ;

        c1_soma <= (out_slv_o1 and (not out_slv_o0)) ;
        c2_soma <= out_slv_o0 ;


--=====================================
--    Descricao do MUX de ENTRADA
--=====================================

        with (out_slv_cont(1) & out_slv_cont(0)) select
                out_mux_entrada0(0 to 15) <= not entrada(0 to 15) when B"00" ,
                                             out_slv_rsped01(0 to 15)  when others ;

        with (out_slv_cont(1) & out_slv_cont(0)) select
                out_mux_entrada1(0 to 15) <= not entrada(0 to 15) when B"01" ,
                                             out_slv_rsped01(16 to 31)  when others ;

        with (out_slv_cont(1) & out_slv_cont(0)) select
                out_mux_entrada2(0 to 15) <= not entrada(0 to 15) when B"10" ,
                                             out_slv_rp(0 to 15)  when others ;

        with (out_slv_cont(1) & out_slv_cont(0)) select
                out_mux_entrada3(0 to 15) <= not entrada(0 to 15) when B"11" ,
                                             out_slv_rp(16 to 31)  when others ;


--=================================
--    Descricao do MUX de SAIDA
--=================================


        with out_slv_o0 select
                out_mux_saida <= out_slv_rsped45 when '0',
                                 soma   when '1' ;

        with sel_out select
```

```
                saida(0 to 15) <=  not out_mux_saida(0 to 15)  when '0',
                                   not out_mux_saida(16 to 31) when '1' ;

end comportamental ;
```

```
--============================
--      VETORES DE TESTES
--============================

in      entrada (15 downto 0) B;
in      sel_out B;
in      o0 B;
in      o1 B;
in      load B;
in      reset B;
in      opera B;
in      ckm B;
in      cks B;
out     saida (15 downto 0) B;
out     fim B;
out     carryout B;
in      vdd B;
in      vss B;

# pattern list

begin

-- Pattern description :

--    e                 sool ro  cc  s                 f  cvv
--    n                 e01oep kk  a                   i  ads
--    t                 l    ase ms  i                 m rds
--    r                 _    der     d                   r
--    a                 o    t a     a                   y
--    d                 u                                o
--    a                 t                                u
--                                                       t
p0  : 000000000000000000000000?11111111111111111?0?110;
p1  : 000000000000000000000010?11111111111111111?0?110;
p2  : 001100110011001101001001?11111111111111111?0?110;
p3  : 001100110011001101001010?11111111111111111?0?110;
p4  : 001100110011001101001001?11111111111111111?1?110;
p5  : 001100110011001101001010?11111111111111111?1?110;
p6  : 001100110011001101001001?11111111111111111?1?110;
p7  : 001100110011001101000010?11111111111111111?1?110;
p8  : 001100110011001101000001?11111111111111111?1?110;
p9  : 001100110011001101000010?11111111111111111?1?110;
p10 : 001100110011001101000001?11111111111111111?1?110;
p11 : 001100110011001101000010?11111111111111111?1?110;
```

p12 : 00110010001100100101000?1111111111111111?1?110;
p13 : 00110010001100100101001?1111111111111111?1?110;
p14 : 00110010001100100101000?0011001000110010?1?110;
p15 : 00110010001100100101001?0011001000110010?1?110;
p16 : 00110010001100100101000?0011001000110010?1?110;
p17 : 00110010001100100101001?0011001000110010?1?110;
p18 : 00110011001100110100000?0011001000110010?1?110;
p19 : 00110011001100110100001?0011001000110010?1?110;
p20 : 00110011001100110100000?0011001000110010?1?110;
p21 : 00110011001100110100001?0011001000110010?1?110;
p22 : 00110011001100110100000?0011001000110010?1?110;
p23 : 00110011001100110100001?0011001000110010?1?110;
p24 : 00110011001100110101000?0011001000110010?1?110;
p25 : 00110011001100110101001?0011001000110010?1?110;
p26 : 00110011001100110101000?0011001000110010?1?110;
p27 : 00110011001100110101001?0011001000110010?1?110;
p28 : 00110011001100110101000?0011001000110010?1?110;
p29 : 00110011001100110101001?0011001000110010?1?110;
p30 : 00110001100000100100000?0011001000110010?1?110;
p31 : 00110001100000100100001?0011001000110010?1?110;
p32 : 00110001100000100100000?0011001000110010?1?110;
p33 : 00110001100000100100001?0011001000110010?1?110;
p34 : 00110001100000100100000?0011001000110010?1?110;
p35 : 00110001100000100100001?0011001000110010?1?110;
p36 : 00110011001100110101000?0011001000110010?1?110;
p37 : 00110011001100110101001?0011001000110010?1?110;
p38 : 00110011001100110101000?0110010101100110?1?110;
p39 : 00110011001100110101001?0110010101100110?1?110;
p40 : 00110011001100110101000?0110010101100110?1?110;
p41 : 00110011001100110101001?0110010101100110?1?110;
p42 : 00110011001100110100000?0110010101100110?1?110;
p43 : 00110011001100110100001?0110010101100110?1?110;
p44 : 00110011001100110100000?0110010101100110?1?110;
p45 : 00110011001100110100001?0110010101100110?1?110;
p46 : 00110011001100110101000?0110010101100110?1?110;
p47 : 00110011001100110101001?0110010101100110?1?110;
p48 : 00110011001100110101000?0110010101100110?1?010;
p49 : 00110011001100110101001?0110010101100110?1?010;
p50 : 00110011001100110101000?0110010101100110?1?010;
p51 : 00110011001100110101001?0110010101100110?1?010;
p52 : 00110011001100110100000?0110010101100110?1?010;
p53 : 00110011001100110100001?0110010101100110?1?010;
p54 : 00110011001100110100010?0110010101100110?1?010;
p55 : 00110011001100110100011?0110010101100110?1?010;
p56 : 00110011001100110100010?0110010101100110?1?010;
p57 : 00110011001100110100011?0110010101100110?1?010;
p58 : 00110011001100110100000?0110010101100110?1?010;
p59 : 00110011001100110100001?0110010101100110?1?010;
p60 : 00110011001100110100000?0110010101100110?1?010;
p61 : 00110011001100110100001?0110010101100110?1?010;
p62 : 00110011001100111100000?0110011001100110?1?010;
p63 : 00110011001100111100001?0110011001100110?1?010;
p64 : 00110011001100111100000?0110011001100110?1?010;
p65 : 00110011001100111100001?0110011001100110?1?010;
p66 : 00110011001100110100000?0110010101100110?1?010;
p67 : 00110011001100110100001?0110010101100110?1?010;
p68 : 00110011001100110100000?0110010101100110?1?010;
p69 : 00110011001100110100001?0110010101100110?1?010;

p70 : 0011001100110011010000010?0110010101100110?1?010;
p71 : 0011001100110011010000010?0110010101100110?1?010;
p72 : 0011001100110011010000010?0110010101100110?1?010;
p73 : 0011001100110011010000010?0110010101100110?1?010;
p74 : 0011001100110011010000010?0110010101100110?1?010;
p75 : 0011001100110011010000010?0110010101100110?1?010;
p76 : 0011001100110011010000010?0110010101100110?1?010;
p77 : 0011001100110011010000010?0110010101100110?1?010;
p78 : 0011001100110011010000010?0110010101100110?1?010;
p79 : 0011001100110011010000010?0110010101100110?1?010;
p80 : 0011001100110011011000010?0110010101100110?1?010;
p81 : 0011001100110011011000010?0110010101100110?1?010;
p82 : 0011001100110011011001010?0110010101100110?1?010;
p83 : 0011001100110011011001100?0110010101100110?1?010;
p84 : 0011001100110011011001010?1111111101111110?1?010;
p85 : 0011001100110011011001100?1111111101111110?1?010;
p86 : 0011001100110011011000010?1111111101111110?1?010;
p87 : 0011001100110011011000100?1111111101111110?1?010;
p88 : 0011001100110011011000010?1111111101111110?1?010;
p89 : 0011001100110011011000100?1111111101111110?1?010;
p90 : 0011001100110011011000010?1111111101111110?1?010;
p91 : 0011001100110011011000100?1111111101111110?1?010;
p92 : 0011001100110011011000010?1111111101111110?1?010;
p93 : 0011001100110011011000100?1111111101111110?1?010;
p94 : 0011001100110011111000010?1111111111111111?1?010;
p95 : 0011001100110011111000100?1111111111111111?1?010;
p96 : 0011001100110011111000010?1111111111111111?1?010;
p97 : 0011001100110011111000100?1111111111111111?1?010;
p98 : 0011001100110011111000010?1111111111111111?1?010;
p99 : 0011001100110011011000100?1111111101111110?1?010;
p100 : 0011001100110011011000010?1111111101111110?1?010;
p101 : 0011001100110011000000100?1111111101111110?1?010;
p102 : 0011001100110011000000010?1111111101111110?1?010;
p103 : 0011001100110011000000100?1111111101111110?1?010;
p104 : 0011001100110011000000010?1111111101111110?1?010;
p105 : 0011001100110011000000100?1111111101111110?1?010;
p106 : 0011001100110011000010010?1111111101111110?1?010;
p107 : 0011001100110011000010100?1111111101111110?1?010;
p108 : 0011001100110011000010010?1111111111111111?1?110;
p109 : 0011001100110011000010100?1111111111111111?1?110;
p110 : 0011001100110011000010010?1111111111111111?1?110;
p111 : 0011001100110011000000100?1111111111111111?1?110;
p112 : 0011001100110011000000010?1111111111111111?1?110;
p113 : 0011001100110011000000100?1111111111111111?1?110;
p114 : 0011001100110011000100010?1111111111111111?1?110;
p115 : 0011001100110011000100100?1111111111111111?1?110;
p116 : 0011001100110011000100010?0011001100110011?1?110;
p117 : 0011001100110011000100100?0011001100110011?1?110;
p118 : 0011001100110011000100010?0011001100110011?1?110;
p119 : 0011001100110011000000100?0011001100110011?1?110;
p120 : 1111111111111111000000010?0011001100110011?1?110;
p121 : 1111111111111111000000100?0011001100110011?1?110;
p122 : 1111111111111111000100010?0011001100110011?1?110;
p123 : 1111111111111111000100100?0011001100110011?1?110;
p124 : 1111111111111111000100010?0011001100110011?1?110;
p125 : 1111111111111111000100100?0011001100110011?1?110;
p126 : 1111111111111111000100010?0011001100110011?1?110;
p127 : 0011001100110011000000100?0011001100110011?1?110;

p128 : 00110011001100110000000?001100110011001?1?110;
p129 : 00110011001100110000001?001100110011001?1?110;
p130 : 00110011001100110001000?001100110011001?1?110;
p131 : 00110011001100110001001?001100110011001?1?110;
p132 : 00110011001100110001000?011001100110011?1?110;
p133 : 00110011001100110001001?011001100110011?1?110;
p134 : 00110011001100110001000?011001100110011?1?110;
p135 : 11111111111111110000001?011001100110011?1?110;
p136 : 11111111111111110000000?011001100110011?1?110;
p137 : 00110011001100110000001?011001100110011?1?110;
p138 : 11111111111111111001000?111111111111111?1?110;
p139 : 11111111111111111001001?111111111111111?1?110;
p140 : 11111111111111111001000?111111111111111?1?110;
p141 : 11111111111111111001001?111111111111111?1?110;
p142 : 11111111111111111001000?111111111111111?1?110;
p143 : 00110011001100111000001?111111111111111?1?110;
p144 : 00110011001100111000000?111111111111111?1?110;
p145 : 00110011001100111000001?111111111111111?1?110;
p146 : 00110011001100111000101?111111111111111?1?110;
p147 : 00110011001100111000110?111111111111111?1?110;
p148 : 00110011001100111000101?111111111111111?0?110;
p149 : 00110011001100111000110?111111111111111?0?110;
p150 : 00110011001100111000101?111111111111111?0?110;
p151 : 00110011001100111000001?111111111111111?0?110;
p152 : 00110011001100111000000?111111111111111?0?110;
p153 : 00110011001100111000001?111111111111111?0?110;
p154 : 00110011001100111000000?111111111111111?0?110;
p155 : 00110011001100111000001?111111111111111?0?110;
p156 : 00110011001100111000000?111111111111111?0?110;
p157 : 00110011001100111000001?111111111111111?0?110;
p158 : 00110011001100111000000?111111111111111?0?110;
p159 : 00110011001100111000001?111111111111111?0?110;
p160 : 00110011001100111000000?100110011001100?0?110;
p161 : 00110011001100111000001?100110011001100?0?110;
p162 : 00110011001100111000000?110011001100100?0?110;
p163 : 00110011001100111000001?110011001100100?0?110;
p164 : 00110011001100111000000?011001100110010?0?110;
p165 : 00110011001100111000001?011001100110010?0?110;
p166 : 00110011001100111000000?101100110011001?0?110;
p167 : 00110011001100111000001?101100110011001?0?110;
p168 : 00110011001100111000000?101100110011001?0?110;
p169 : 00110011001100111000001?101100110011001?0?110;
p170 : 00110011001100111000000?110110011001100?0?110;
p171 : 00110011001100111000001?110110011001100?0?110;
p172 : 00110011001100111000000?110110011001100?0?110;
p173 : 00110011001100111000001?110110011001100?0?110;
p174 : 00110011001100111000000?111011001100100?0?110;
p175 : 00110011001100111000001?111011001100100?0?110;
p176 : 00110011001100111000000?100001100110010?0?110;
p177 : 00110011001100111000001?100001100110010?0?110;
p178 : 00110011001100111000000?110000110011001?0?110;
p179 : 00110011001100111000001?110000110011001?0?110;
p180 : 00110011001100111000000?010111001100101?0?110;
p181 : 00110011001100111000001?010111001100101?0?110;
p182 : 00110011001100111000000?101011100110010?0?110;
p183 : 00110011001100111000001?101011100110010?0?110;
p184 : 00110011001100111000000?101011100110010?0?110;
p185 : 00110011001100111000001?101011100110010?0?110;

p186 : 00110011001100111000000l?110101110011001l?0?110;
p187 : 00110011001100111000001O?110101110011001l?0?110;
p188 : 00110011001100111000000l?110101110011001l?0?110;
p189 : 00110011001100111000001O?110101110011001l?0?110;
p190 : 00110011001100111000000l?111010111001100l?0?110;
p191 : 00110011001100111000001O?111010111001100l?0?110;
p192 : 00110011001100111000000l?100001010011001l?0?110;
p193 : 00110011001100111000001O?100001010011001l?0?110;
p194 : 00110011001100111000000l?110000101001100l?0?110;
p195 : 00110011001100111000001O?110000101001100l?0?110;
p196 : 00110011001100111000000l?010111000011001l?0?110;
p197 : 00110011001100111000001O?010111000011001l?0?110;
p198 : 00110011001100111000000l?101011100001100l?0?110;
p199 : 00110011001100111000001O?101011100001100l?0?110;
p200 : 00110011001100111000000l?101011100001100l?0?110;
p201 : 00110011001100111000001O?101011100001100l?0?110;
p202 : 00110011001100111000000l?110101110000110O?0?110;
p203 : 00110011001100111000001O?110101110000110O?0?110;
p204 : 00110011001100111000000l?110101110000110O?0?110;
p205 : 00110011001100111000001O?110101110000110O?0?110;
p206 : 00110011001100111000000l?111010111000011O?0?110;
p207 : 00110011001100111000001O?111010111000011O?0?110;
p208 : 00110011001100111000000l?100001010010000O?0?110;
p209 : 00110011001100111000001O?100001010010000O?0?110;
p210 : 00110011001100111000000l?110000101001000O?0?110;
p211 : 00110011001100111000001O?110000101001000O?0?110;
p212 : 00110011001100111000000l?010111000010101O?1?110;
p213 : 00110011001100111000001O?010111000010101O?1?110;
p214 : 00110011001100111000000l?010111000010101O?1?110;
p215 : 00110011001100111000001O?010111000010101O?1?110;
p216 : 00110011001100111000000l?010111000010101O?1?110;
p217 : 00110011001100111000001O?010111000010101O?1?110;
p218 : 00110011001100111000000l?010111000010101O?1?110;
p219 : 00110011001100111000001O?010111000010101O?1?110;
p220 : 00110011001100111000000l?010111000010101O?1?110;
p221 : 00110011001100111000001O?010111000010101O?1?110;
p222 : 00110011001100110000000l?001111010110111l?1?110;
p223 : 00110011001100110000001O?001111010110111l?1?110;
p224 : 00110011001100110000000l?001111010110111l?1?110;
p225 : 00110011001100110000001O?001111010110111l?1?110;
p226 : 00110011001100110000000l?001111010110111l?1?110;
p227 : 00110011001100110000001O?001111010110111l?1?110;
p228 : 00110011001100111010100l?010111000010101O?1?110;
p229 : 00110011001100111010101O?010111000010101O?1?110;
p230 : 00110111001100111010100l?11111111111111111?1?010;
p231 : 11111111111111111010101O?11111111111111111?1?010;
p232 : 11111111111111111010100l?11111111111111111?1?010;
p233 : 11111111111111111010101O?11111111111111111?1?010;
p234 : 11111111111111111010100l?11111111111111111?1?010;
p235 : 11111111111111111010001O?11111111111111111?1?010;
p236 : 00100011000110101011000l?11111111111111111?1?010;
p237 : 00100011000110101011001O?11111111111111111?1?010;
p238 : 00100011000110101011000l?11111111111111111?1?010;
p239 : 00100011000110101011001O?11111111111111111?1?010;
p240 : 00100011000110101011000l?11111111111111111?1?010;
p241 : 00100011000110101011001O?11111111111111111?1?010;
p242 : 10111111111111101010000l?11111111111111111?1?010;
p243 : 10111111111111101010001O?11111111111111111?1?010;

p244 : 1011111111111111010100001?111111111111111111?1?010;
p245 : 1011111111111111010100010?111111111111111111?1?010;
p246 : 0011110100010000101100017111111111111111111?1?010;
p247 : 0011110100010000101100107111111111111111111?1?010;
p248 : 0011110100010000101100017001111010001000?1?010;
p249 : 0011110100010000101100107001111010001000?1?010;
p250 : 0011110100010000101100017001111010001000?1?010;
p251 : 0011110100010000101100107001111010001000?1?010;
p252 : 1011111111111011010100001?001111010001000?1?010;
p253 : 1011111111111011010100010?001111010001000?1?010;
p254 : 1011111111111011010100001?001111010001000?1?010;
p255 : 1011111111111011010100010?001111010001000?1?010;
p256 : 1010111000010100101100017001111010001000?1?010;
p257 : 1010111000010100101100107001111010001000?1?010;
p258 : 1010111000010100101100017001111010001000?1?010;
p259 : 1010111000010100101100107001111010001000?1?010;
p260 : 1010111000010100101100017001111010001000?1?010;
p261 : 1010111000010100101100107001111010001000?1?010;
p262 : 1010111000010000101000017001111010001000?1?010;
p263 : 1011111111010000101000107001111010001000?1?010;
p264 : 1011111111010000101000017001111010001000?1?010;
p265 : 1011111111010000101000107001111010001000?1?010;
p266 : 1111111100111000101100017001111010001000?1?010;
p267 : 1111111100111000101100107001111010001000?1?010;
p268 : 1111111100111000101100017001111011101011?1?010;
p269 : 1111111100111000101100107001111011101011?1?010;
p270 : 1111111100111000101100017001111011101011?1?010;
p271 : 1111111100111000101100107001111011101011?1?010;
p272 : 1111111111111100101000017001111011101011?1?010;
p273 : 1111111111111100101000107001111011101011?1?010;
p274 : 1111111111111100101001017001111011101011?1?010;
p275 : 1111111111111100101001107001111011101011?1?010;
p276 : 1111111111111100101001017111111111111111?0?110;
p277 : 1111111111111100101001107111111111111111?0?110;
p278 : 1111111111111100101001017111111111111111?0?110;
p279 : 1111111111111100101000107111111111111111?0?110;
p280 : 1111111111111100101000017111111111111111?0?110;
p281 : 1111111111111100101000107111111111111111?0?110;
p282 : 1111111111111100101000017111111111111111?0?110;
p283 : 1111111111111100101000107111111111111111?0?110;
p284 : 1111111111111100101000017111111111111111?0?110;
p285 : 1111111111111100101000107111111111111111?0?110;
p286 : 1111111111111100101000017111111111111111?0?110;
p287 : 1111111111111100101000107111111111111111?0?110;
p288 : 1111111111111100101000017111111111111111?0?110;
p289 : 1111111111111100101000107111111111111111?0?110;
p290 : 1111111111111100101000017111111111111111?0?110;
p291 : 1111111111111100101000107111111111111111?0?110;
p292 : 1111111111111100101000017111111111111111?0?110;
p293 : 1111111111111100101000107111111111111111?0?110;
p294 : 1111111111111100101000017111111111111111?0?110;
p295 : 1111111111111100101000107111111111111111?0?110;
p296 : 1111111111111100101000017111111111111111?0?110;
p297 : 1111111111111100101000107111111111111111?0?110;
p298 : 1111111111111100101000017111111111111111?0?110;
p299 : 1111111111111100101000107111111111111111?0?110;
p300 : 1111111111111100101000017111111111111111?0?110;
p301 : 1111111111111100101000107111111111111111?0?110;

p360 : 1111111111111100101000017?1111111111111111?1?110;
p361 : 1111111111111100101000107?1111111111111111?1?110;
p362 : 1111111111111100101000017?1111111111111111?1?110;
p363 : 1111111111111100101000107?1111111111111111?1?110;
p364 : 1111111111111100101000017?1111111111111111?1?110;
p365 : 1111111111111100101000107?1111111111111111?1?110;
p366 : 1111111111111100101000017?1111111111111111?1?110;
p367 : 1111111111111100101000107?1111111111111111?1?110;
p368 : 1111111111111100101000017?1111111111111111?1?110;
p369 : 1111111111111100101000107?1111111111111111?1?110;
p370 : 1111111111111100101000017?1111111111111111?1?110;
p371 : 1111111111111100101000107?1111111111111111?1?110;
p372 : 1111111111111100101000017?1111111111111111?1?110;
p373 : 1111111111111100101000107?1111111111111111?1?110;
p374 : 1111111111111100101000017?1111111111111111?1?110;
p375 : 1111111111111100101000107?1111111111111111?1?110;
p376 : 1111111111111100101000017?1111111111111111?1?110;
p377 : 1111111111111100101000107?1111111111111111?1?110;
p378 : 1111111111111100101000017?1111111111111111?1?110;
p379 : 1111111111111100101000107?1111111111111111?1?110;
p380 : 1111111111111100001000017?1111111100000101?1?110;
p381 : 1111111111111100001000107?1111111100000101?1?110;
p382 : 1111111111111100001000017?1111111100000101?1?110;
p383 : 1111111111111100001000107?1111111100000101?1?110;
p384 : 1111111111111100101001017?1111111111111111?1?110;
p385 : 1111111111111100101001107?1111111111111111?1?110;
p386 : 1111111111111100101001017?1111111111111111?0?110;
p387 : 1111111111111100101001107?1111111111111111?0?110;
p388 : 1111111111111100101001017?1111111111111111?0?110;
p389 : 1111111111111100101001107?1111111111111111?0?110;
p390 : 1111111111111100101001017?1111111111111111?0?110;
p391 : 1111111111111100101000107?1111111111111111?0?110;
p392 : 1111111111111100101000017?1111111111111111?0?010;
p393 : 1111111111111100101000107?1111111111111111?0?010;
p394 : 1111111111111100101000017?1111111111111111?0?110;
p395 : 1111111111111100101000107?1111111111111111?0?110;
p396 : 1111111111111100101000017?1111111111111111?0?110;
p397 : 1111111111111100101000107?1111111111111111?0?110;
p398 : 1111111111111100101000017?1111111111111111?0?010;
p399 : 1111111111111100101000107?1111111111111111?0?010;
p400 : 1111111111111100101000017?1111111111111111?0?110;
p401 : 1111111111111100101000107?1111111111111111?0?110;
p402 : 1111111111111100101000017?1111111111111111?0?010;
p403 : 1111111111111100101000107?1111111111111111?0?010;
p404 : 1111111111111100101000017?1111111111111111?0?110;
p405 : 1111111111111100101000107?1111111111111111?0?110;
p406 : 1111111111111100101000017?1111111111111111?0?010;
p407 : 1111111111111100101000107?1111111111111111?0?010;
p408 : 1111111111111100101000017?1111111111111111?0?110;
p409 : 1111111111111100101000107?1111111111111111?0?110;
p410 : 1111111111111100101000017?1111111111111111?0?010;
p411 : 1111111111111100101000107?1111111111111111?0?010;
p412 : 1111111111111100101000017?1111111111111111?0?110;
p413 : 1111111111111100101000107?1111111111111111?0?110;
p414 : 1111111111111100101000017?1111111111111111?0?110;
p415 : 1111111111111100101000107?1111111111111111?0?110;
p416 : 1111111111111100101000017?1111111111111111?0?010;

p418 : 1111111111111100101000017111111111111111107?7110;
p419 : 1111111111111100101000107111111111111111107?7110;
p420 : 1111111111111100101000017111111111111111107?7010;
p421 : 1111111111111100101000107111111111111111107?7010;
p422 : 1111111111111100101000017111111111111111007?7110;
p423 : 1111111111111100101000107111111111111111007?7110;
p424 : 1111111111111100101000017111111111111111007?7110;
p425 : 1111111111111100101000107111111111111111007?7110;
p426 : 1111111111111100101000017111111111111111007?7010;
p427 : 1111111111111100101000107111111111111111007?7010;
p428 : 1111111111111100101000017111111111111110007?7110;
p429 : 1111111111111100101000107111111111111110007?7110;
p430 : 1111111111111100101000017111111111111110007?7010;
p431 : 1111111111111100101000107111111111111110007?7010;
p432 : 1111111111111100101000017111111111111100007?7110;
p433 : 1111111111111100101000107111111111111100007?7110;
p434 : 1111111111111100101000017111111111111100007?7010;
p435 : 1111111111111100101000107111111111111100007?7010;
p436 : 1111111111111100101000017111111111111000017?7110;
p437 : 1111111111111100101000107111111111111000017?7110;
p438 : 1111111111111100101000017111111111111000017?7010;
p439 : 1111111111111100101000107111111111111000017?7010;
p440 : 1111111111111100101000017111111111111000107?7110;
p441 : 1111111111111100101000107111111111111000107?7110;
p442 : 1111111111111100101000017111111111111000107?7110;
p443 : 1111111111111100101000107111111111111000107?7110;
p444 : 1111111111111100101000017111111111111000101?7010;
p445 : 1111111111111100101000107111111111111000101?7010;
p446 : 1111111111111100101000017111111111111000101?7110;
p447 : 1111111111111100101000107111111111111000101?7110;
p448 : 1111111111111100101000017111111111111000101?7110;
p449 : 1111111111111100101000107111111111111000101?7110;
p450 : 1111111111111100101000017111111111111000010?7110;
p451 : 1111111111111100101000107111111111111000010?7110;
p452 : 1111111111111100101000017111111111111000010?7010;
p453 : 1111111111111100101000107111111111111000010?7010;
p454 : 1111111111111100101000017111111111111000010?7110;
p455 : 1111111111111100101000107111111111111000010?7110;
p456 : 1111111111111100101000017111111111111000010?7010;
p457 : 1111111111111100101000107111111111111000010?7010;
p458 : 1111111111111100101000017111111111111000010?7110;
p459 : 1111111111111100101000107111111111111000010?7110;
p460 : 1111111111111100101000017111111111111000010?7010;
p461 : 1111111111111100101000107111111111111000010?7010;
p462 : 1111111111111100101000017111111111111000010?7110;
p463 : 1111111111111100101000107111111111111000010?7110;
p464 : 1111111111111100101000017111111111111000010?7110;
p465 : 1111111111111100101000107111111111111000010?7110;
p466 : 1111111111111100101000017111111111111000010?7110;
p467 : 1111111111111100101000107111111111111000010?7110;
p468 : 1111111111111100101000017111111111111000010?7110;
p469 : 1111111111111100101000107111111111111000010?7110;
p470 : 1111111111111100101000017111111111111000010?7010;
p471 : 1111111111111100101000107111111111111000010?7010;
p472 : 1111111111111100101000017111111111111000010?7110;
p473 : 1111111111111100101000107111111111111000010?7110;
p474 : 1111111111111100101000017111111111111000010?7110;
p475 : 1111111111111100101000107111111111111000010?7110;

```
p476 : 1111111111111100101000017001011010000100170?110;
p477 : 1111111111111100101000107001011010000100170?110;
p478 : 1111111111111100101000017010110100001001070?110;
p479 : 1111111111111100101000107010110100001001070?110;
p480 : 1111111111111100101000017101101000010010070?010;
p481 : 1111111111111100101000107101101000010010070?010;
p482 : 1111111111111100101000017011010000100100070?110;
p483 : 1111111111111100101000107011010000100100070?110;
p484 : 1111111111111100101000017011010000100100070?110;
p485 : 1111111111111100101000107011010000100100070?110;
p486 : 1111111111111100101000017110100001001000070?010;
p487 : 1111111111111100101000107110100001001000070?010;
p488 : 1111111111111100101000017101000010010000171?110;
p489 : 1111111111111100101000107101000010010000171?110;
p490 : 1111111111111100101000017101000010010000171?110;
p491 : 1111111111111100101000107101000010010000171?110;
p492 : 1111111111111100101000017101000010010000171?110;
p493 : 1111111111111100101000107101000010010000171?110;
p494 : 1111111111111100101000017101000010010000171?110;
p495 : 1111111111111100101000107101000010010000171?110;
p496 : 1111111111111100001000017011000111011101071?110;
p497 : 1111111111111100001000107011000111011101071?110;
p498 : 1111111111111100001000017011000111011101071?110;
p499 : 1111111111111100001000107011000111011101071?110;
p500 : 1111111111111100001000017011000111011101071?110;
end;
```

BIBLIOGRAFIA

[MOR90] MORAES, Misael Elias - Microprocessor-based three fase transduter, IEEE Instrumentation and Medition - p. 98-132, 1990

[HWA79] HWANG, Kai - Computer arithmetic principles: principles, architecture and design - Ed. Wiley, 1979

[SCA85] SCANLON, LEO J. - IBM PC & XT ASSEMBLY LANGUAGE: A Guide For Programmers Enhanced and Enlarged - Ed. BRADY, 1985

[RBS67] ROTH, J. P. ; BOURICUS, W. G. AND SCHNEIDER, P. R. - PROGRAMED ALGORITHMS TO COMPUTE TESTS TO DETECT AND DISTINGUISH BETWEEN FAILURES IN LOGIC CIRCUITS - IEEE Trans. on Electronic Computers, vol. EC-16, pp. 547-580, 1967

[IEEE88] IEEE - IEEE Standard VHDL/1076 Reference Manual: The Institute of Electric and Electronic Engineers - 1988

[TAU82] TAUB, Herbert & SHILLING, Donald- Eletrônica Digital - Ed. McGraw Hill, 1982

[MAS93a] MASI/CAO & VLSI - Université Pierre et Marrie Curie - SCLIB: a portable CMOS standard cell library - Paris, jan/1993

[MAS93b] MASI/CAO & VLSI - Université Pierre et Marrie Curie - Asimut: a simulation tool for hardware description - Paris, jan/1993

[MAS93c] MASI/CAO & VLSI - Université Pierre et Marrie Curie - VST, VBE: supported structural VHDL subset - Paris, jan/1993

[MAS93d] MASI/CAO & VLSI - Université Pierre et Marrie Curie - SCR: standard cell router - Paris, jan/1993

[MAS93e] MASI/CAO & VLSI - Université Pierre et Marrie Curie - Ring: pad ring router - Paris, jan/1993

[MAS93f] MASI/CAO & VLSI - Université Pierre et Marrie Curie - Genlib: Procedural Design Language Based Upon C - Paris, jan/1993

[MAS93g] MASI/CAO & VLSI - Université Pierre et Marrie Curie - Lvx: Logical Versus eXtracted Net-List Comparator - Paris, jan/1993

[MAS93h] MASI/CAO & VLSI - Université Pierre et Marrie Curie - Lynx: Hierarchical Net-list extractor - Paris, jan/1993

[MAS93i] MASI/CAO & VLSI - Université Pierre et Marrie Curie - Proof: Formal Proof Between Two Behavioural Descriptions - Paris, jan/1993

[MAS93j] MASI/CAO & VLSI - Université Pierre et Marrie Curie - S2R: Creates a Real Layout Cell from a Symbolic Layout Cell and a Tecnology File - Paris, jan/1993

[MAS93k] MASI/CAO & VLSI - Université Pierre et Marrie Curie - Versatil: Design Rule Checker - Paris, jan/1993

[MAS93l] MASI/CAO & VLSI - Université Pierre et Marrie Curie - ALC: ALLIANCE Hierarchical Symbolic Layout Editor - Paris, jan/1993

[MAS93m] MASI/CAO & VLSI - Université Pierre et Marrie Curie - Desb: Functional Abstraction of CMOS Circuits - Paris, jan/1993

[MAS93n] MASI/CAO & VLSI - Université Pierre et Marrie Curie - PADLIB: PAD Cells Library - Paris, jan/1993

[MAS93o] MASI/CAO & VLSI - Université Pierre et Marrie Curie - CATAL: Catalog File Format - Paris, jan/1993